Systême soft

Программный комплекс Систэм Платформ

# SePlatform.Development Studio 1.2

## Руководство пользователя

| Редакция | Соответствует версии ПО |
| --- | --- |
| 3. Предварительная | 1.2.3 |

# Содержание

# 1. Назначение и возможности

Настоящее руководство пользователя предназначено для специалистов по разработке, внедрению и эксплуатации АСУ любого масштаба – от локальных до территориально распределенных систем.

Руководство содержит полное описание функциональности среды разработки SePlatform.Development Studio, предназначенной для разработки проектов АСУ на базе Систэм Платформ и проектов развертывания, с привязкой созданных АСУ к местам исполнения.

SePlatform.Development Studio – компонент Систэм Платформ, предназначенный для разработки проектов автоматизации и их внедрения на объекте. SePlatform.Development Studio позволяет автоматизировать процессы создания, сборки и размещения проекта приложения посредством получения полной информации о структуре и составе приложения, а также о привязках к среде исполнения.

SePlatform.Development Studio позволяет синхронно формировать и модифицировать несколько проектов внутри одного решения. Разработка нескольких обособленных проектов может осуществляться одновременно при запуске нескольких экземпляров SePlatform.Development Studio. Основными структурными единицами проекта являются исходные файлы проекта, содержащие описание всех элементов модели автоматизируемых объектов и объектов среды исполнения.

С помощью SePlatform.Development Studio возможно выполнение следующих действий:

- описание физической структуры объектов автоматизации;
- описание логической структуры объектов автоматизации применительно к средствам автоматизации различной функциональной направленности (сервера сбора данных, сервера истории, сервера межуровневого транспорта);
- представление схемы развертывания проекта автоматизации на вычислительных средствах автоматизируемых объектов;
- проведение дистрибуции (распространения) конфигураций проекта на средства автоматизации проекта в рамках схемы развертывания.

Существует возможность разработки проекта приложения несколькими людьми. Этому способствует разделение разрабатываемого проекта на несколько модулей.

# 2. Установка и удаление

## 2.1. Системные требования

| | |
|---|---|
| ОС | Microsoft Windows 10 Pro/11 Pro<br>Microsoft Windows Server 2012/2012 R2/2016/2019/2022 |
| Разрядность ОС | x64 |
| Процессор | Intel Celeron с тактовой частотой не менее 1.6 ГГц |
| Объем оперативной памяти | не менее 2 ГБ |
| Объем дисковой памяти | не менее 1 ГБ |
| Сетевой адаптер | Ethernet 10/100/1000 Мбит/с. |
| Установленное ПО | › Антивирусное ПО<br>› Microsoft Visual C++ 2015-2019 (x64) Redistributable<br>https://learn.microsoft.com/ru-ru/cpp/windows/latest-supported-vc-redist?view=msvc-170<br>› Microsoft .NET Framework 4.6.1<br>https://www.microsoft.com/en-US/download/details.aspx?id=49982 |

ОБРАТИТЕ ВНИМАНИЕ

Установка и работа SePlatform.Development Studio в ОС Linux не поддерживается. Однако возможно установить и запустить SePlatform.Development Studio под Wine. Ниже приведена инструкция, как это сделать. Корректная работа программы при этом не гарантируется.

## 2.2. ОС Windows

Чтобы установить или удалить SePlatform.Development Studio, запустите установочный файл `*.msi`. Откроется окно мастера установки. Для установки или удаления следуйте инструкциям мастера.

SePlatform.Development Studio устанавливается в папку: `C:\Program Files\SePlatform\SePlatform.Development Studio`.

## 2.3. ОС Linux

ПРИМЕЧАНИЕ

В данной инструкции описана установка и запуск SePlatform.Development Studio в ОС AstraLinux.

### 2.3.1. Установка

1. Добавьте репозитории, необходимые для работы:

1.1. В файл `/etc/apt/sourcelist` добавьте строчки:

```
deb https://dl.astralinux.ru/astra/frozen/2.12_x86-64/2.12.45/repository stable main
contrib non-free
deb [trusted=yes] https://mirror.yandex.ru/debian/ buster main contrib non-free
```

1.2. Перезапустите компьютер.

Репозитории нужны для того, чтобы не возникало проблем со скачиванием необходимых элементов при установке и запуске Wine и Winetricks.

2. Установите необходимое дополнительное ПО:

```
sudo apt update
sudo apt install ia32-libs
sudo apt-get install cabextract
```

3. Установите Wine:

```
wget https://nextcloud.astralinux.ru/s/sp4RioDdyPL6Ayf/download/wine_7.13-0-
astra-se17_amd64.deb
sudo dpkg -i wine_7.13-0-astra-se17_amd64.deb
```

4. Установите Winetricks.

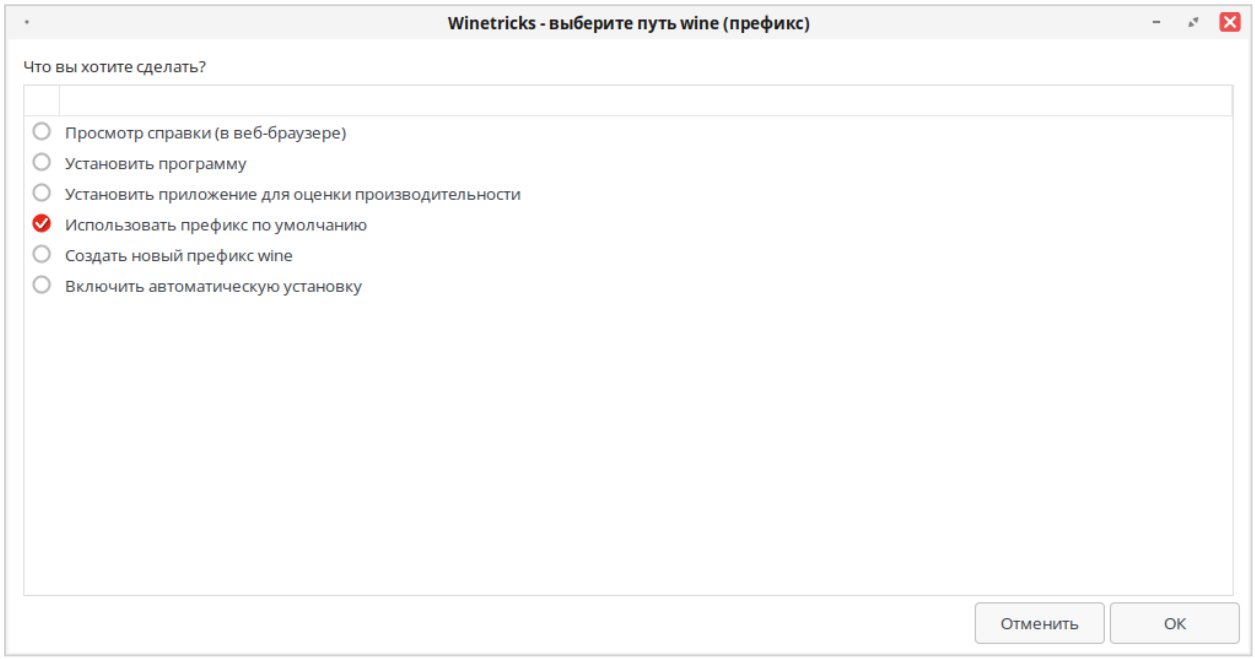
```
wget
https://raw.githubusercontent.com/Winetricks/winetricks/master/src/winetricks
chmod +x winetricks
sudo mv winetricks /usr/bin
```

5. В терминале запустите Winetricks:

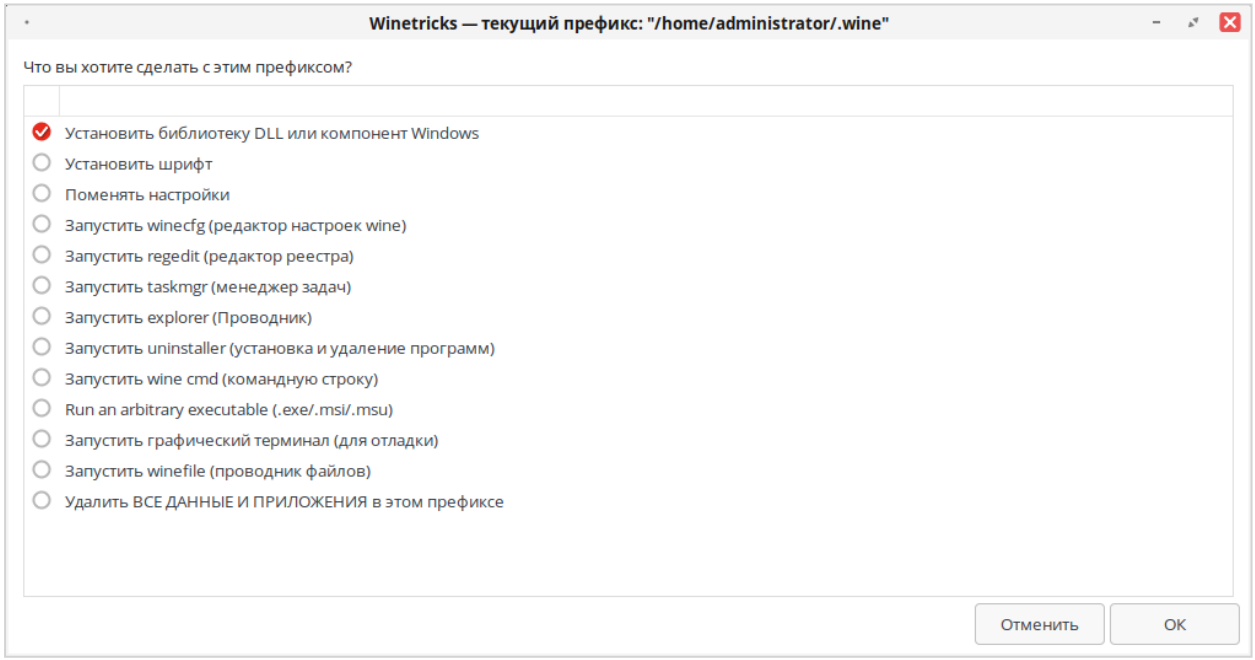
```
winetricks
```

Откроется окно Winetricks. В нём:

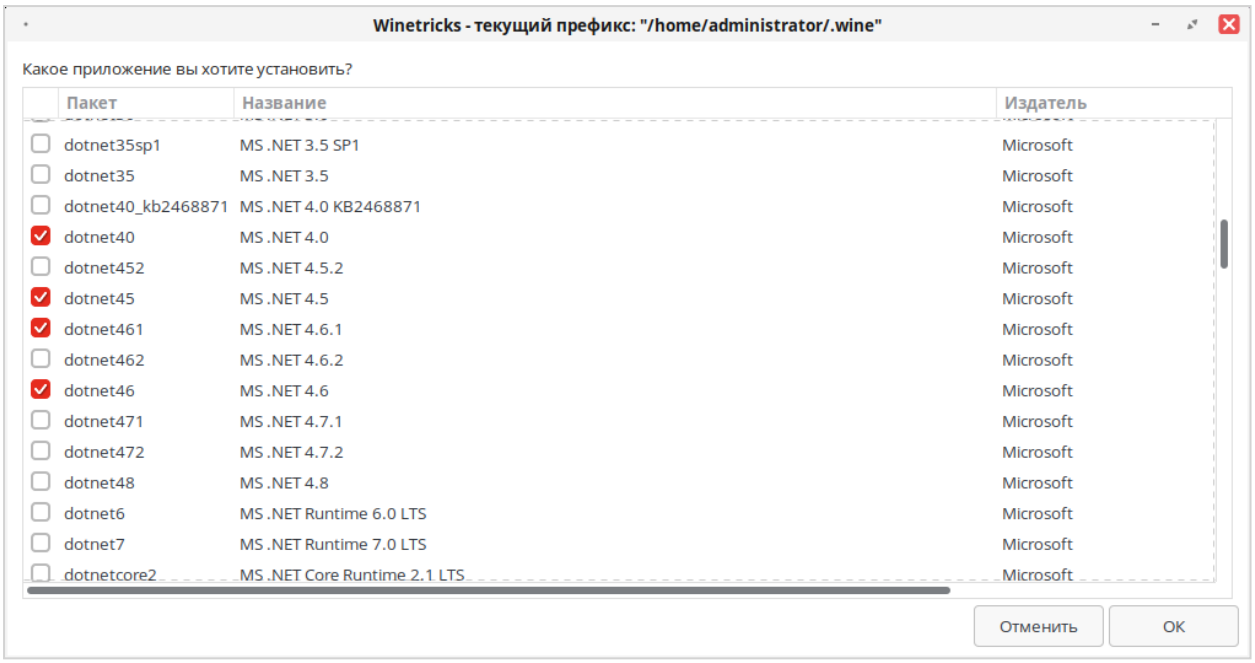
5.1. Выберите актуальный префикс или создайте новый.

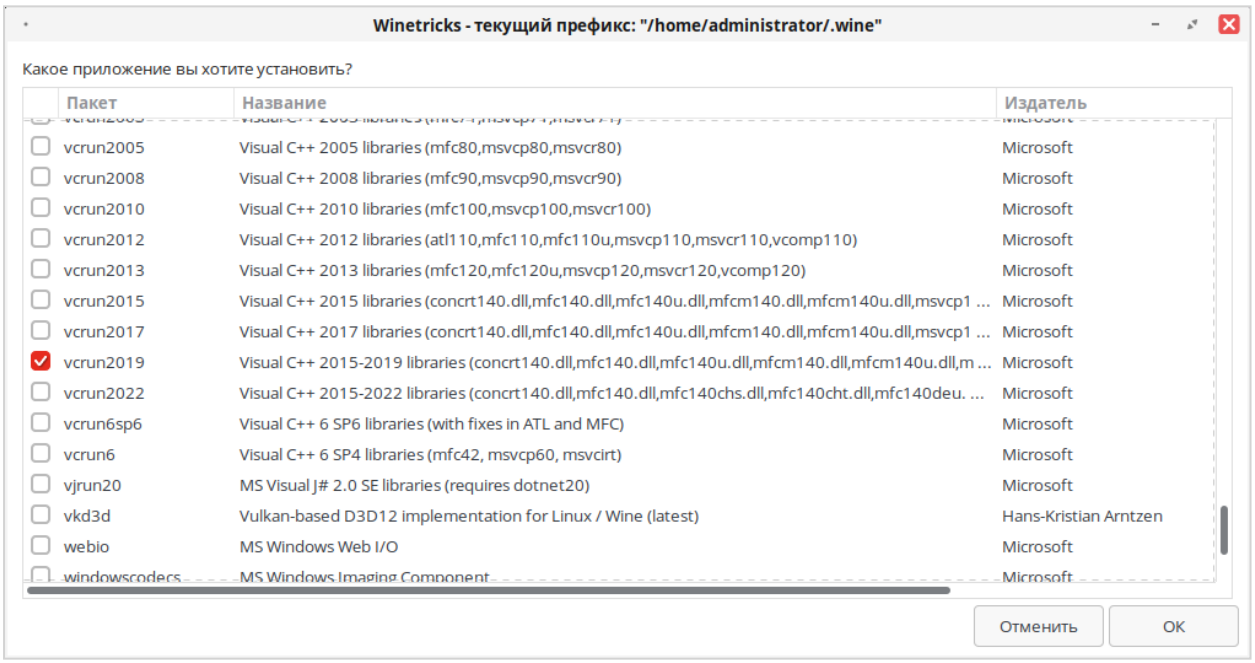



5.2. Перейдите к выбору библиотек DLL.



5.3. Выберите dotnet и vcrun.



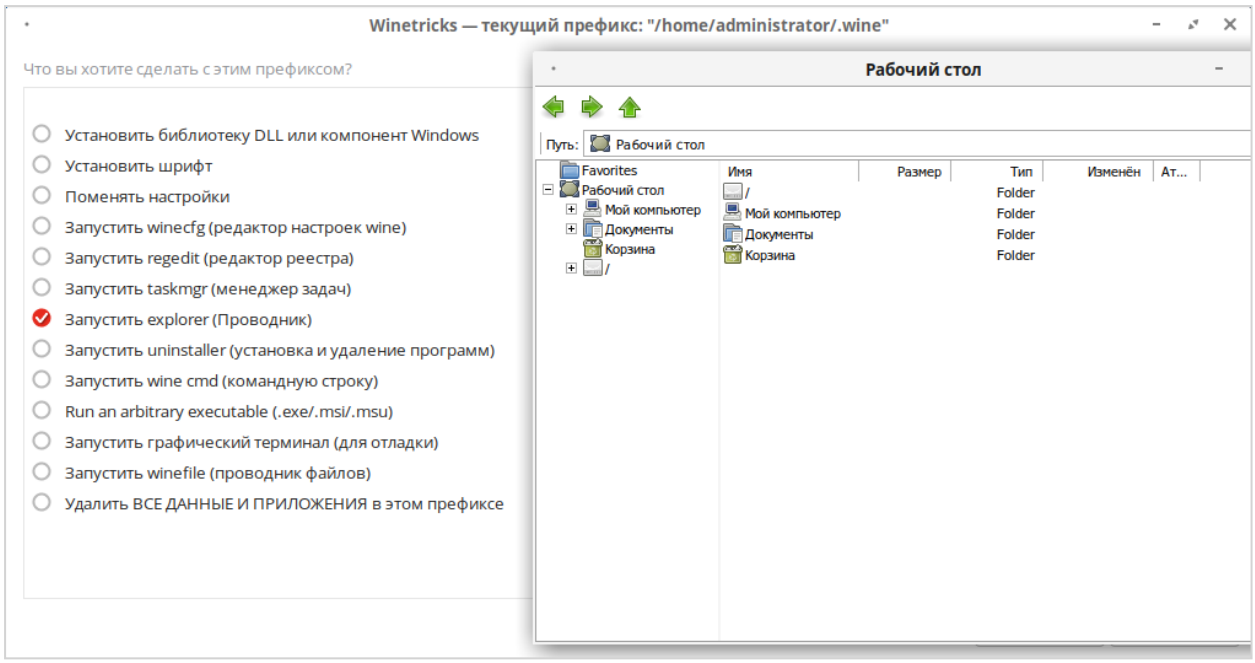




5.4. Установите выбранные компоненты.

ПРИМЕЧАНИЕ

Возникающие в процессе установки предупреждения можно игнорировать, на работоспособность SePlatform.Development Studio они не повлияют.

5.5. Запустите проводник.




5.6. Запустите msi-файл установки SePlatform.Development Studio.

> ОБРАТИТЕ ВНИМАНИЕ
>
> В мастере установки откажитесь от установки хранилища SVN.

## 2.3.2. Запуск

1. Если терминал был закрыт или перезапущен, выполните:

```
export WINE=/opt/wine-7.13/bin/wine
export WINEPREFIX=~/DevStudio
```

> ОБРАТИТЕ ВНИМАНИЕ
>
> Эти команды нужно выполнять каждый раз при запуске терминала (командной строки), чтобы команда Wine работала именно с директорией WINEPREFIX.

2. Выполните команду:

```
wine explorer – Мой компьютер – C:\Program
Files\SePlatform\SePlatform.Development Studio\DevStudio.exe
```

Если SePlatform.Development Studio не запускается:

- Проверьте, включён ли в настройках Wine виртуальный рабочий стол.
- Проверьте, что ядро Linux подходит для работы Wine и SePlatform.Development Studio.

Например, для работы AstraLinux 1.7.4 нужна версия ядра 5.15.

## 2.3.3. Использование SePlatform.Development Studio

Чтобы использовать SePlatform.Development Studio без установки SePlatform.Data Server и SePlatform.Domain, в настройках SePlatform.Development Studio укажите параметры удалённой рабочей среды: меню **Файл** → **Параметры**, в окне выберите узел **Рабочая среда** и в нём укажите параметры удалённого компьютера, который является центральным узлом домена.

Чтобы использовать компьютер SePlatform.Development Studio в качестве центрального узла домена, нужно установить на него SePlatform.Data Server и SePlatform.Domain (устанавливать требуется непосредственно в ОС Linux, а не под Wine).

После выполнения описанных выше настроек, использовать SePlatform.Development Studio можно также, как в ОС Windows.

## 2.3.4. Известные ошибки и проблемы

В SePlatform.Development Studio, запущенной под Wine не отображаются стандартные ошибки. Вместо этого SePlatform.Development Studio прекращает работу с ошибкой от Wine.

# 3. Знакомство с SePlatform.Development Studio

Эта глава посвящена знакомству с работой в SePlatform.Development Studio:

- В первом разделе вы научитесь конфигурировать SePlatform.Data Server с помощью SePlatform.Development Studio и освоите основные этапы работы в ней: создание проекта, описание среды исполнения и выполнение развёртывания.
- Во втором разделе вы научитесь описывать передачу данных между SePlatform.Data Server и другими компонентами.

## 3.1. Создание простого проекта

В этом разделе мы

- Создадим проект.
- Опишем в нём SePlatform.Data Server.
- Выполним развёртывание: построим конфигурацию и применим её к SePlatform.Data Server.

### Подготовка: создаём домен

Домен - это среда исполнения, в которой исполняется проект, описанный в SePlatform.Development Studio. Домен состоит из исполняющих компонентов - экземпляров SePlatform.Data Server и SePlatform.AccessPoint.

В нашем примере в домене будет один исполняющий компонент - SePlatform.Data Server.

1. Устанавливаем SePlatform.Data Server.

   SePlatform.Data Server должен быть установлен на компьютере, где установлена SePlatform.Development Studio, или на компьютере, к которому есть доступ по сети.

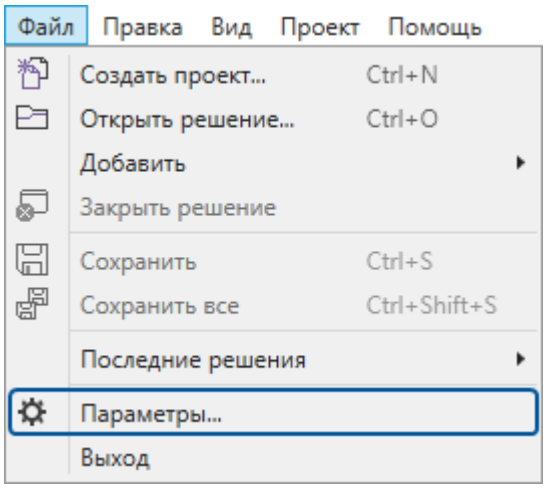
2. Устанавливаем SePlatform.Domain на компьютере, на котором установлен SePlatform.Data Server.

   SePlatform.Domain - это надстройка над исполняющими компонентами, которая объединяет их в домен и позволяет SePlatform.Development Studio их конфигурировать.

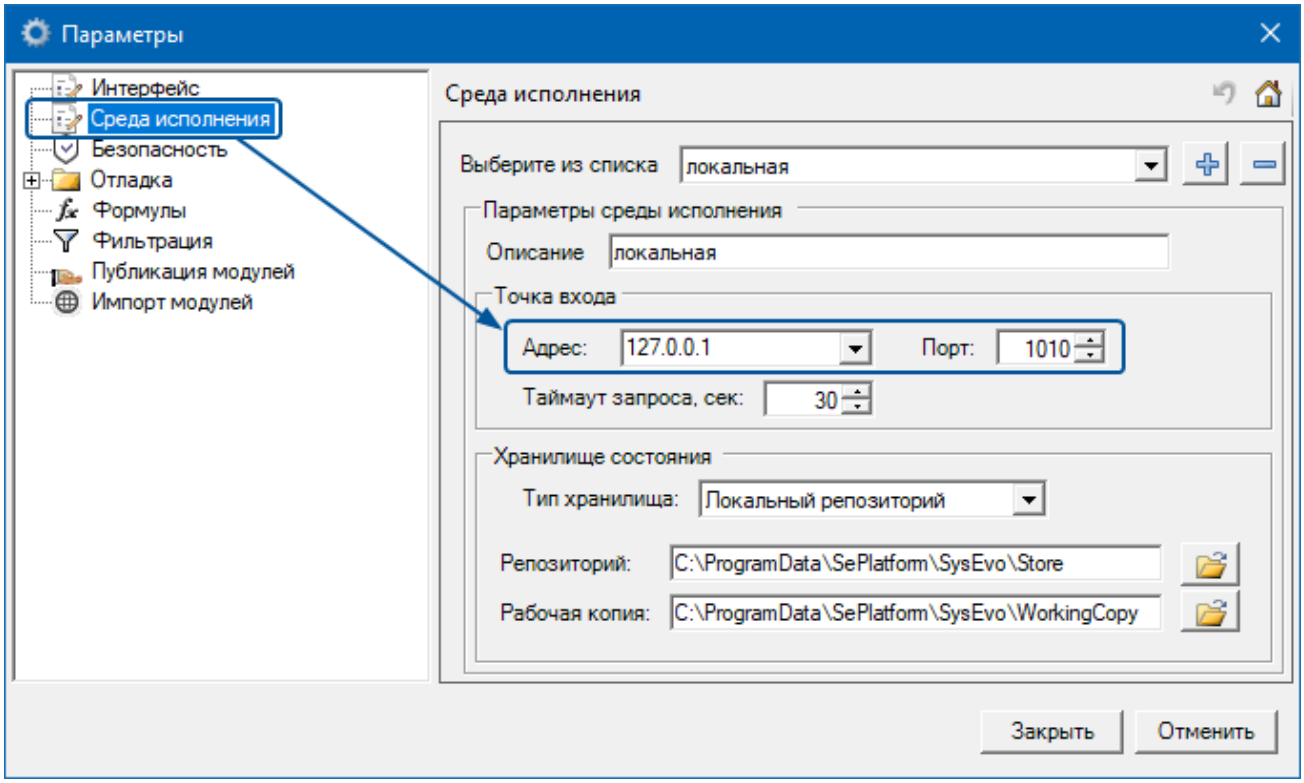
### Настраиваем связь SePlatform.Development Studio с доменом

1. Запускаем SePlatform.Development Studio:
   - В меню Пуск: `Пуск → SePlatform → SePlatform.Development Studio`
   - В файловой системе: `C:\Program Files\SePlatform\SePlatform.Development Studio\DevStudio.exe`

2. Открываем окно параметров: меню **Файл** → **Параметры...**.




3. В узле **Среда исполнения** указываем:
   - IP-адрес компьютера, на котором установлен SePlatform.Domain.
   - Порт доступа к SePlatform.Domain - **«1010»**.



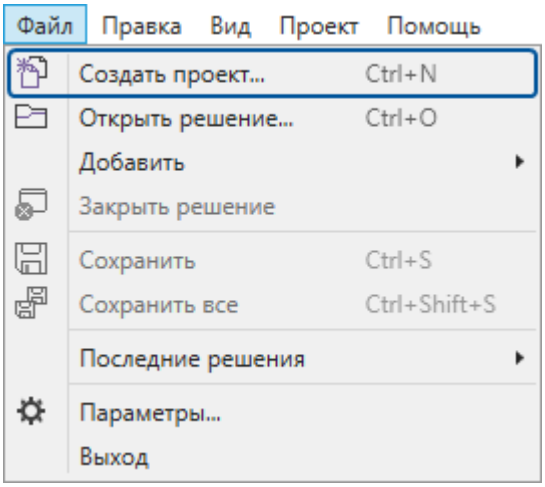

4. Нажимаем кнопку **Закрыть**: изменения будут сохранены.

ПРИМЕЧАНИЕ

Связь с доменом настраивается для приложения, а не для отдельного проекта. Это позволяет развёртывать один и тот же проект в разные домены: рабочий/тестовый.

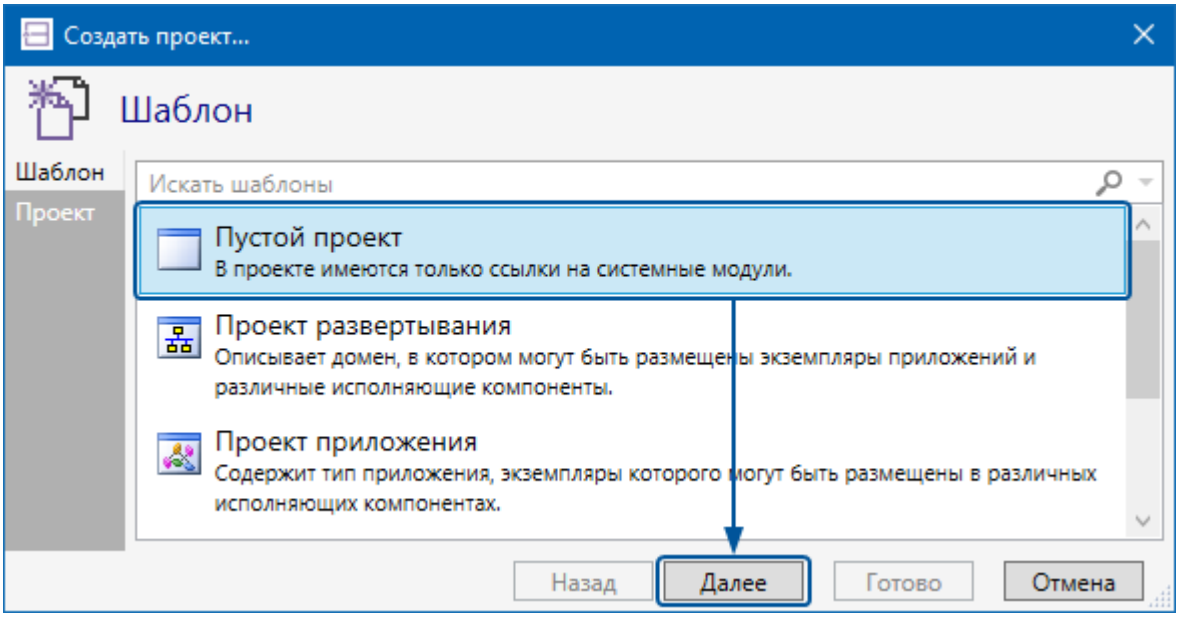
## Создаём проект

1. В меню выбираем: **Файл** → **Создать проект...**.


Файл
Правка
Вид
Проект
Помощь
Создать проект... Ctrl+N
Открыть решение... Ctrl+O
Добавить
Закрыть решение
Сохранить Ctrl+S
Сохранить все Ctrl+Shift+S
Последние решения
Параметры...
Выход

2. В открывшемся окне:

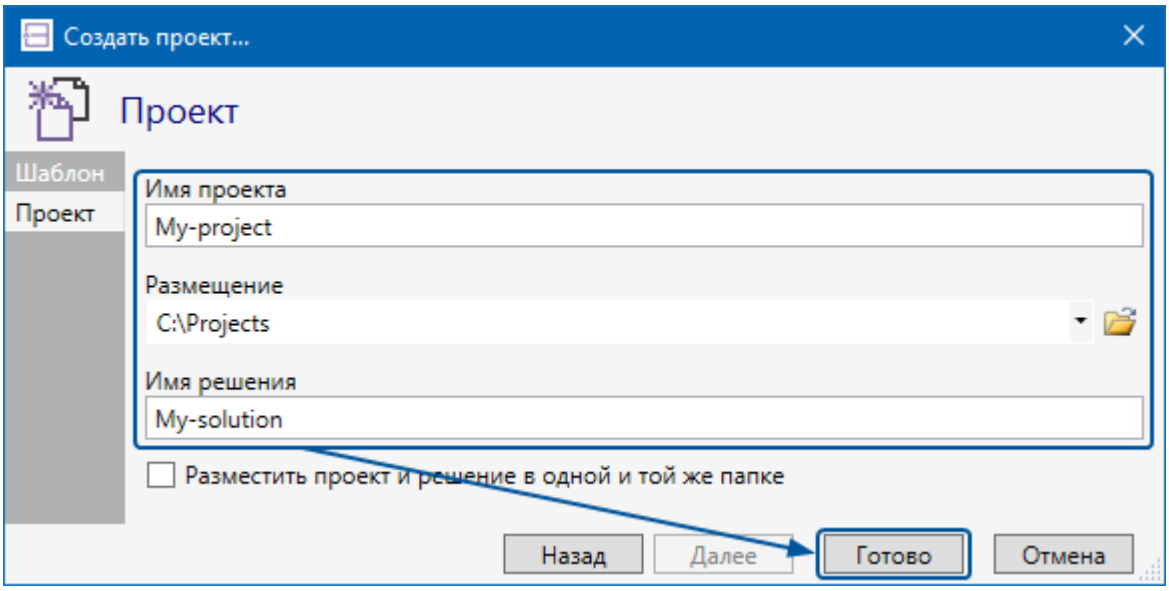
2.1. Выбираем шаблон **Пустой проект** и нажимаем **Далее**.




ПРИМЕЧАНИЕ

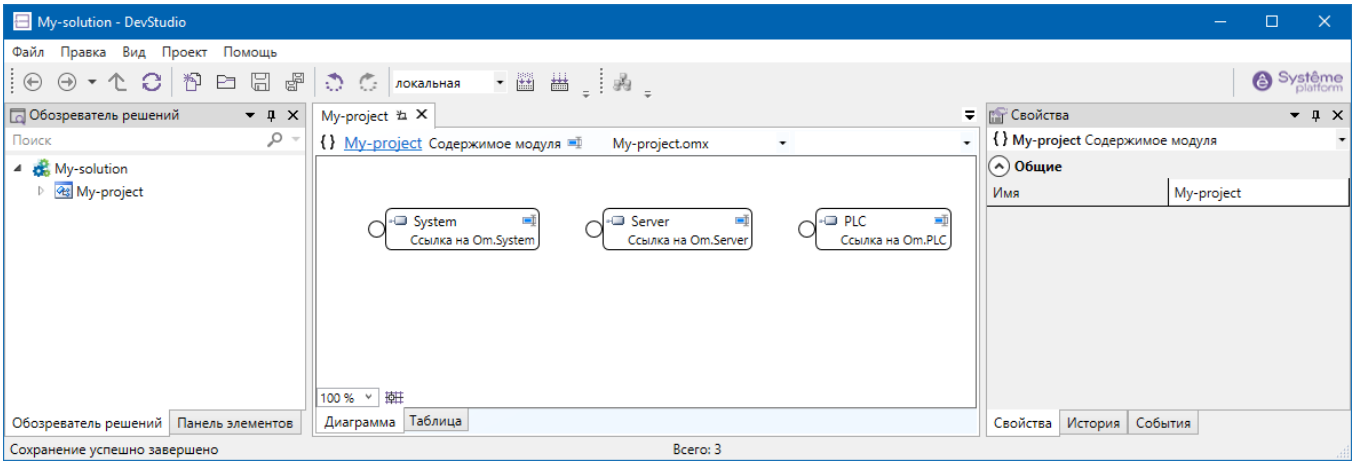
Шаблоны различаются только начальным набором элементов.

2.2. Указываем произвольные имя проекта, папку размещения и имя решения.




В результате:

- В указанной папке будет создано решение.
- Будет создан проект - в указанной папке или в её подпапке в зависимости от того, был ли установлен флаг **Разместить проект и решение в одной и той же папке**.
- Решение откроется в среде разработки.

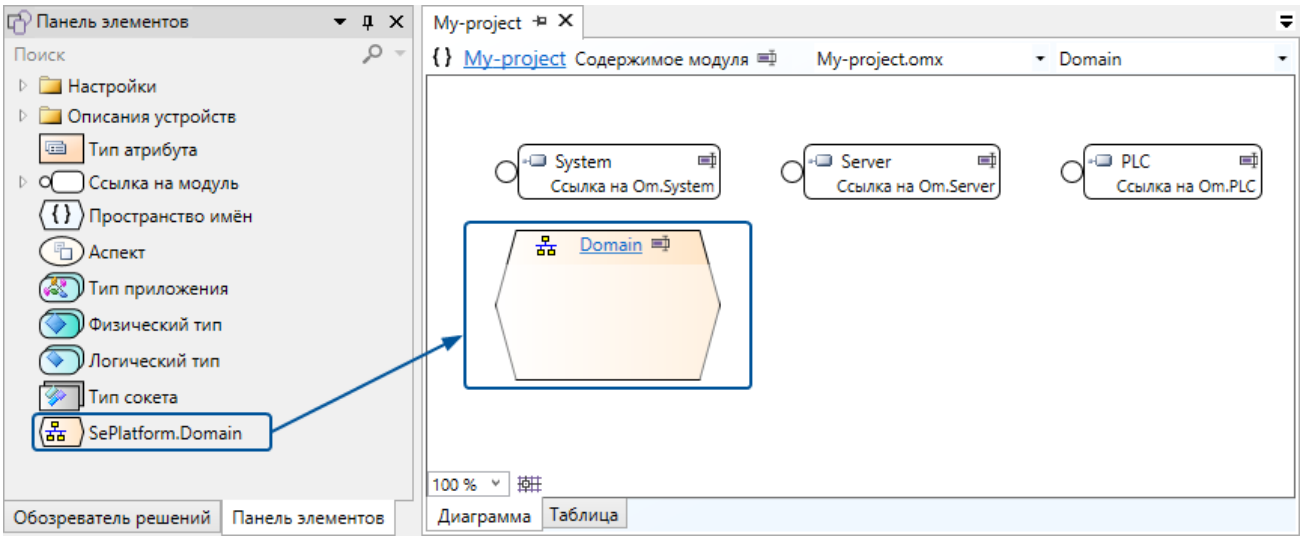



Проект создан. Теперь нужно описать в нём SePlatform.Data Server.

## Описываем SePlatform.Data Server

1. Добавляем в проект элемент **`SePlatform.Domain`**.

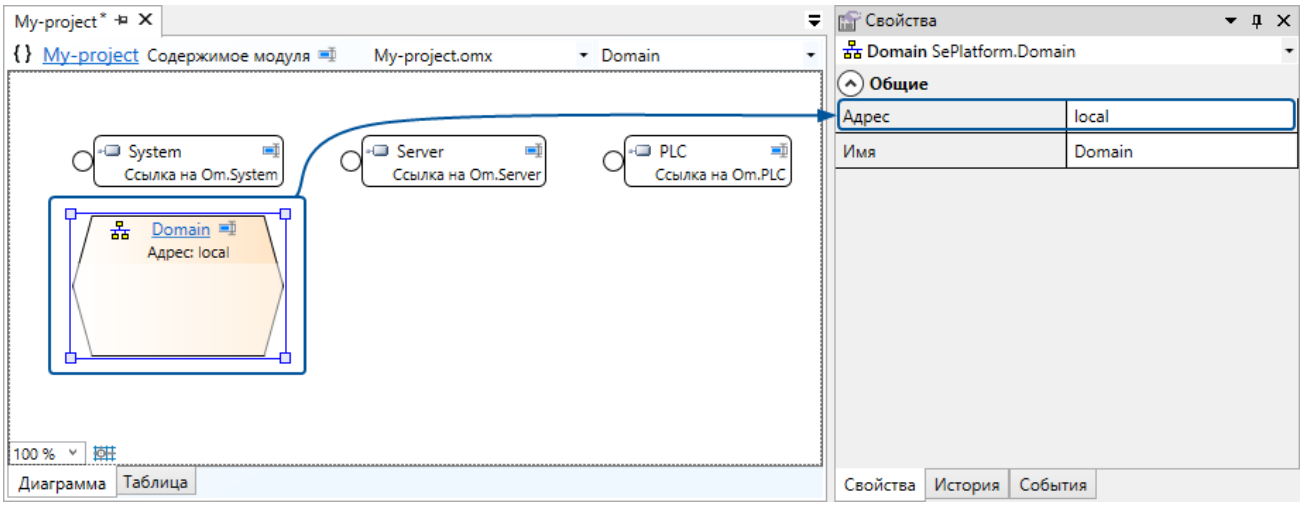
> ПРИМЕЧАНИЕ
>
> Чтобы добавить элемент, перетащите его из панели элементов в рабочую область.




В этом элементе описывается среда исполнения, т.е. домен:

- Компоненты верхнего уровня АСУ ТП: серверные компьютеры, выполняющие сбор, обработку и хранение данных, и АРМы операторов.
- Системы, с которыми они взаимодействуют: компоненты среднего уровня и сторонние системы.

2. Выбираем добавленный элемент и в свойстве **Адрес** указываем **«local»** - имя узла домена, настроенное в SePlatform.Domain.



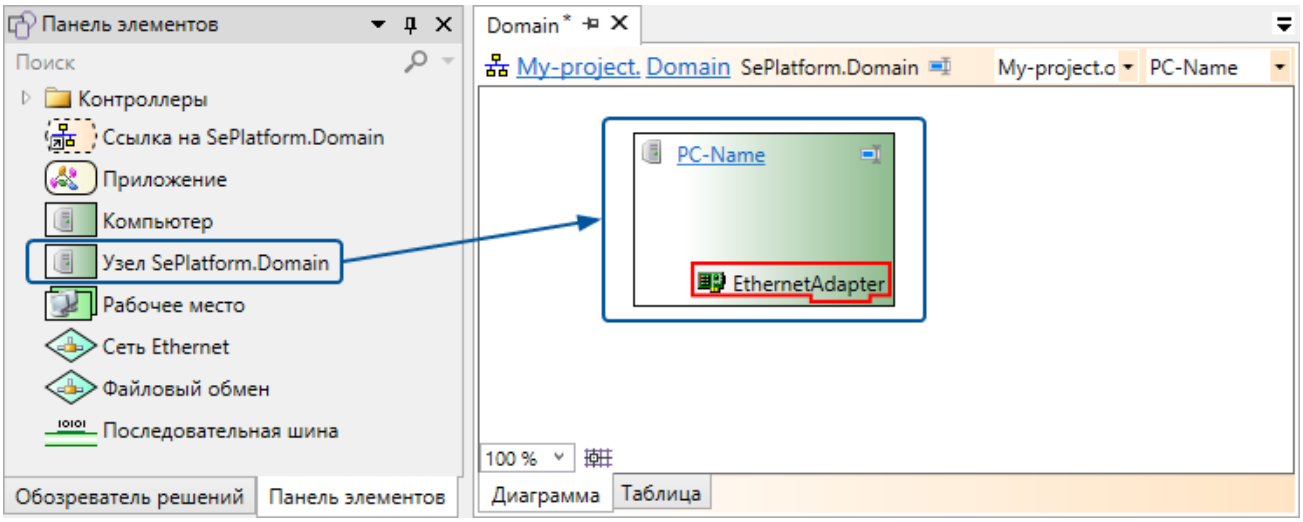

Это адрес центрального узла в домене. Поскольку в нашем домене только один узел (компьютер), он и будет центральным.

3. Переходим в элемент **SePlatform.Domain**.

ПРИМЕЧАНИЕ

Чтобы перейти в элемент, кликните по его имени или дважды кликните по нему в обозревателе решений.

4. Добавляем **Узел SePlatform.Domain**.




**Узел SePlatform.Domain** - это компьютер, на котором есть серверные компоненты Систэм Платформ: SePlatform.Data Server и/или SePlatform.Historian.

В качестве имени указываем сетевое имя компьютера, на котором установлен SePlatform.Data Server.

ОБРАТИТЕ ВНИМАНИЕ

В большинстве случаев элементам можно давать любые имена. Но на узлы домена это правило не распространяется: имя узла домена должно совпадать с сетевым именем компьютера. При создании конфигураций эти имена подставляются в параметры модулей, где используются сетевые имена устройств.

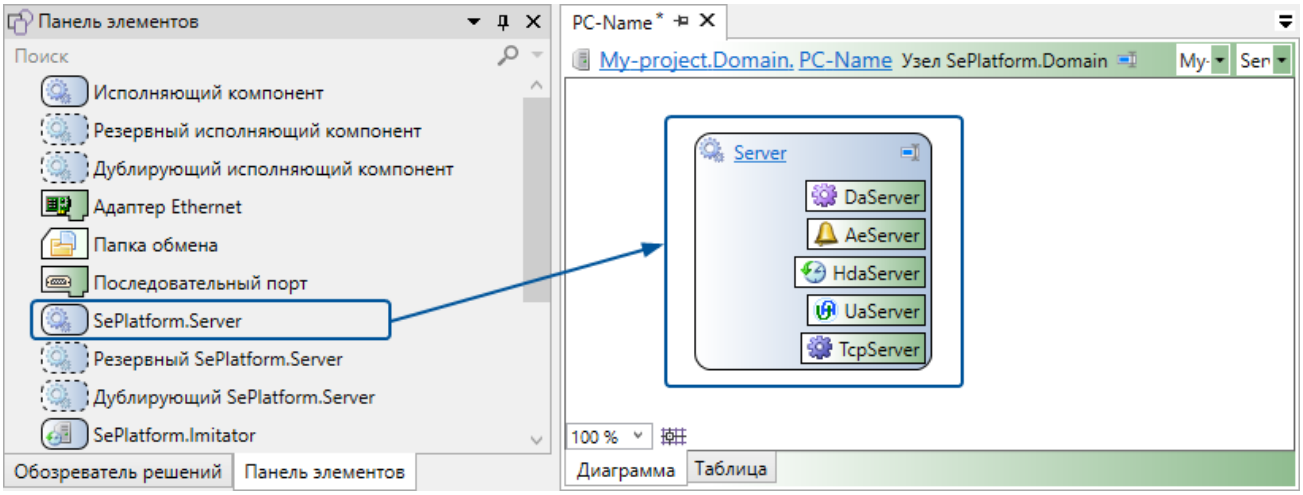
5. Переходим в **Узел SePlatform.Domain**.

6. Удаляем Ethernet-адаптер.

В нашем домене будет только один компонент и обмениваться данными ему не с кем, поэтому Ethernet-адаптер не нужен.

7. Добавляем **`SePlatform.Server`**.

В качестве имени указываем **«`Server`»** - оно должно совпадать с именем, настроенным в SePlatform.Domain.

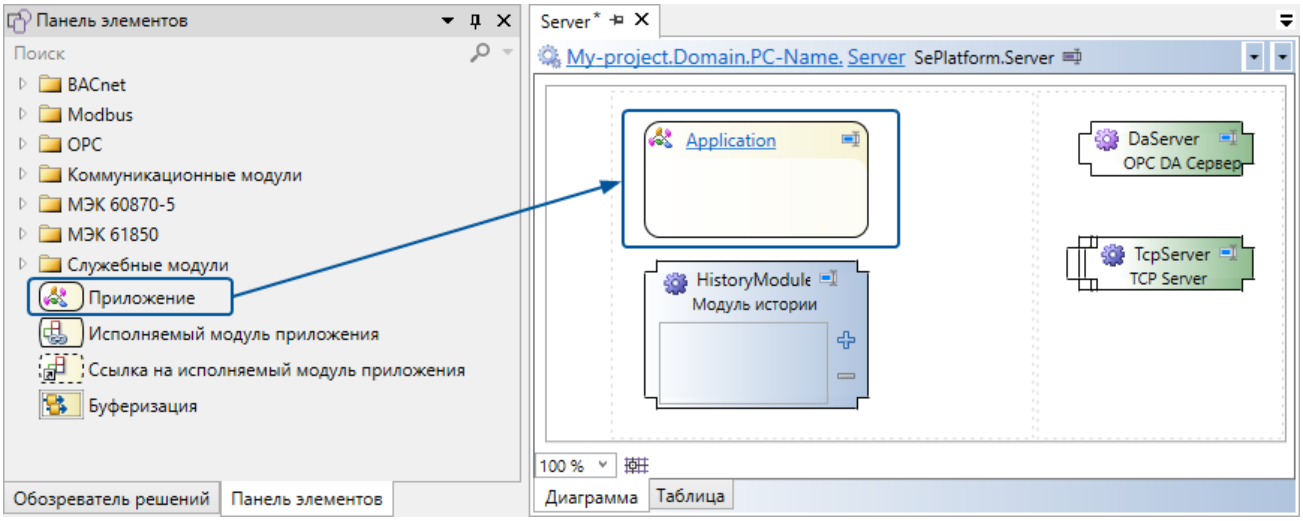



8. Переходим в **`SePlatform.Server`**.

В сервере уже есть начальный набор модулей. Этих модулей для нашего примера будет достаточно.

Теперь надо описать данные сервера.

9. Добавляем **Приложение**.



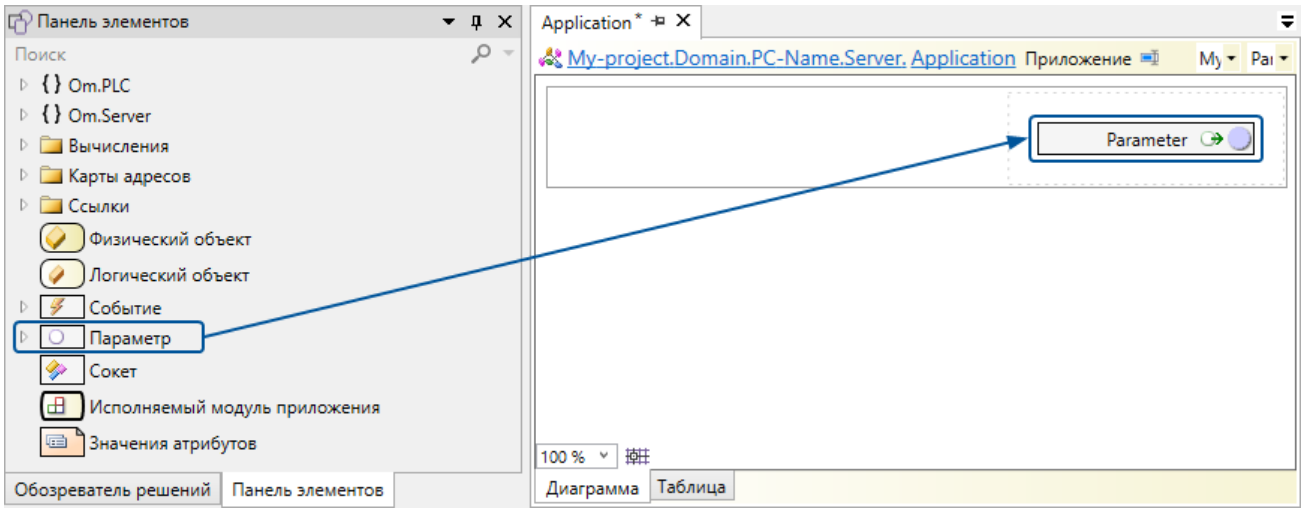

Приложение - это элемент, в котором описываются данные исполняющего компонента. В любом исполняющем компоненте может быть только одно приложение.

10. Переходим в **Приложение**.

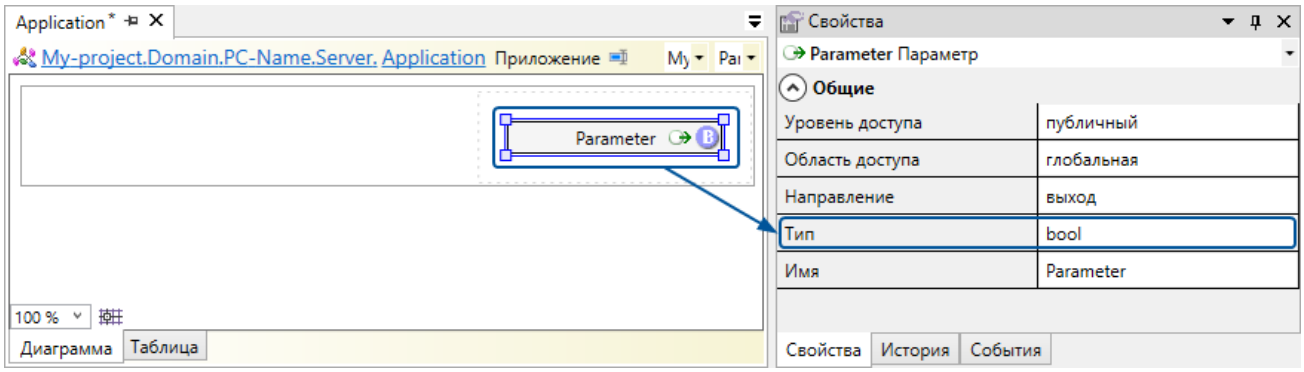
11. Добавляем сигнал.

В SePlatform.Development Studio есть два элемента для описания сигналов: **Параметр** и **Событие**. Оба эти элемента преобразуются в сигналы, но различаются по назначению и функциям, которые выполняют.

Сигнал, который мы добавляем, нужен только для того, чтобы увидеть его в конфигурации; никаких функций он выполнять не будет. Поэтому для описания сигнала можем использовать любой элемент.




12. В свойствах сигнала выбираем любой тип значения.




Для создания конфигурации всё готово.

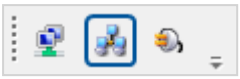
## Выполняем развёртывание

1. Строим решение.

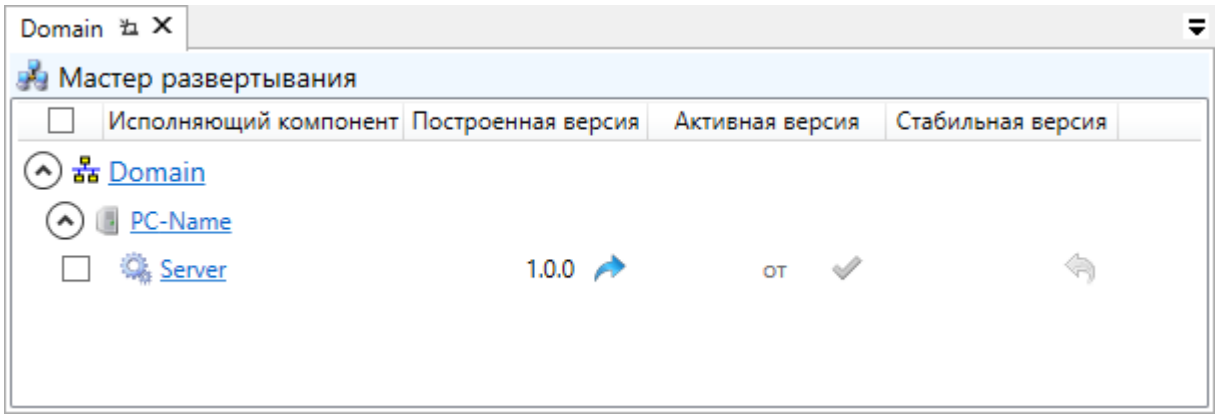

При построении сначала проверяется, что в решении нет ошибок. Если ошибок не обнаружено, создаются конфигурации для исполняющих компонентов.

2. Выполняем развёртывание:

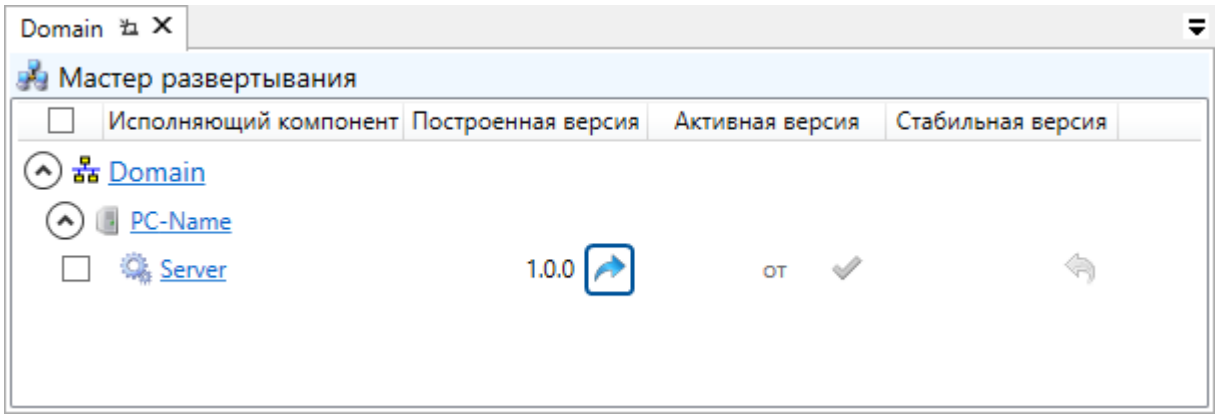
2.1. Открываем **Мастер развёртывания**.



В мастере отображается список исполняющих компонентов, которые SePlatform.Development Studio может конфигурировать. В данном примере в списке будет только один экземпляр SePlatform.Data Server.

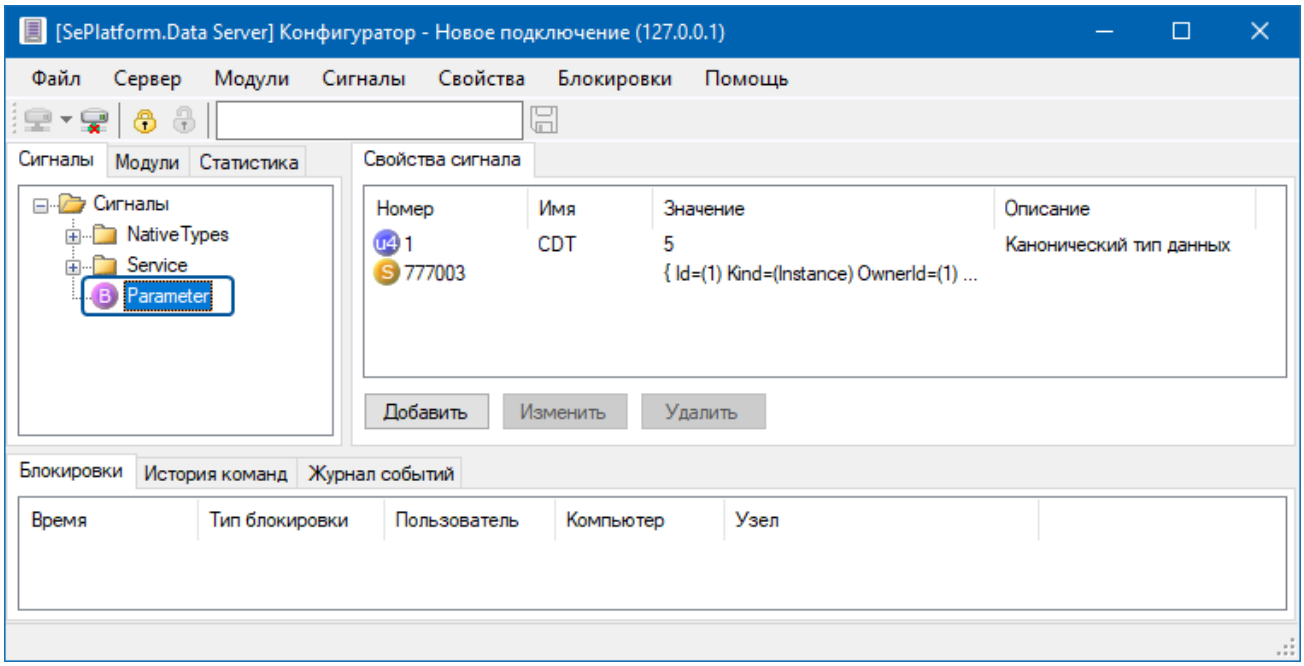



2.2. Применяем конфигурацию к SePlatform.Data Server.




SePlatform.Development Studio передаст конфигурацию компоненту SePlatform.Domain, который применит её к SePlatform.Data Server и перезапустит службу.

3. Проверяем, что конфигурация применилась: подключаемся к SePlatform.Data Server с помощью Конфигуратора или Service - OPCExplorer и убеждаемся, что в дереве сигналов есть добавленный нами сигнал.




# 3.2. Передача данных

Продолжаем работать с проектом, которое создали в предыдущем разделе. В этом разделе мы

- Добавим источник данных (далее - источник).
- Настроим в SePlatform.Data Server сбор данных с источника и передачу ему команд.

## Подготовка: устанавливаем источник

В качестве источника будем использовать эмулятор станции Modbus. Эмулятор можно скачать по ссылке: https://www.modbustools.com/modbus_slave.html. После скачивания, эмулятор нужно установить.

Запускать эмулятор будем на финальном этапе: при проверке, что данные передаются.

## Описываем источник в проекте

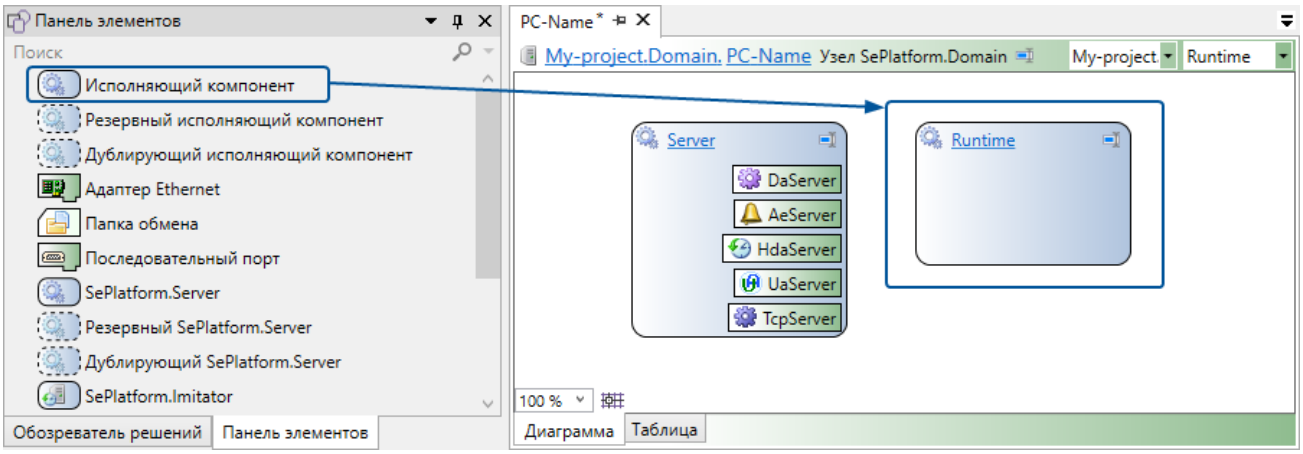
В проекте:

1. Переходим в **`SePlatform.Domain`**.
2. Если эмулятор находится на том же компьютере, что и SePlatform.Data Server, переходим в уже имеющийся **`Узел SePlatform.Domain`**.

   Иначе добавляем **`Компьютер`** и переходим в него. В качестве имени элемента нужно указать сетевое имя компьютера.

   Компьютер - это компьютер или иное сетевое устройство, на котором нет компонентов Систэм Платформ, но с которыми они взаимодействуют.

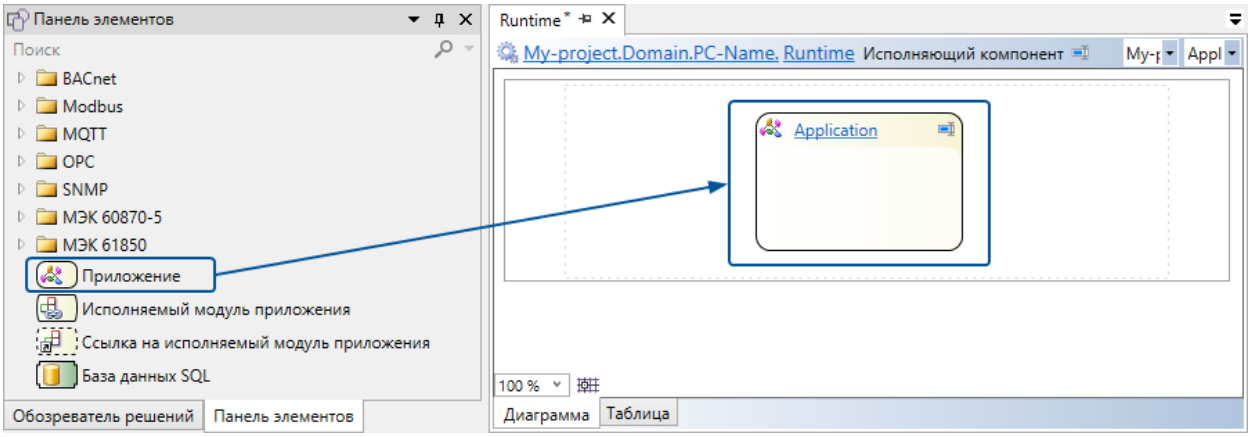
3. Добавляем **Исполняющий компонент** - это будет источник данных.



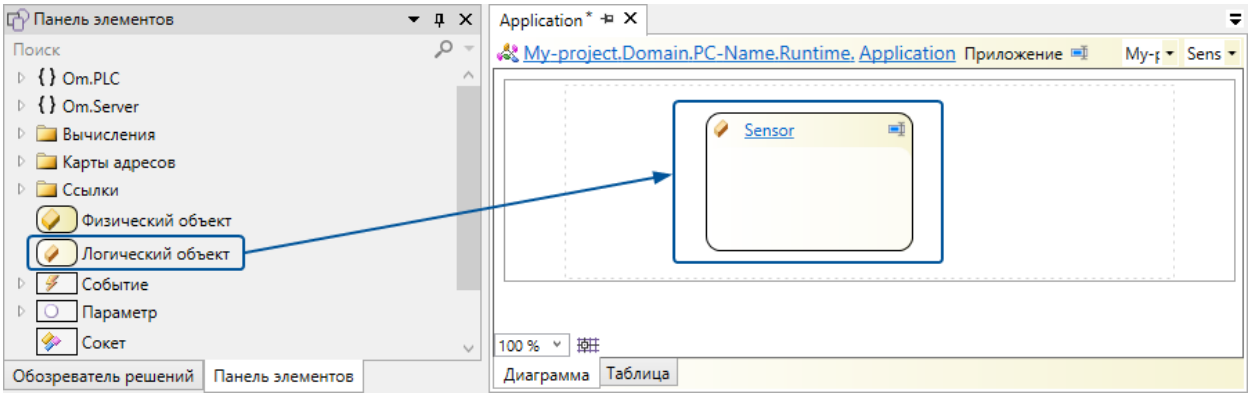

Исполняющий компонент - это любая сторонняя информационная система. Может иметь как программную реализацию (например, служба на компьютере), так и физическую (датчик или другое устройство). SePlatform.Development Studio не конфигурирует сторонние системы, но использует их описание при создании конфигураций для экземпляров SePlatform.Data Server, которые с ними взаимодействуют.

4. Описываем данные источника:

4.1. В **Исполняющий компонент** добавляем **Приложение**.

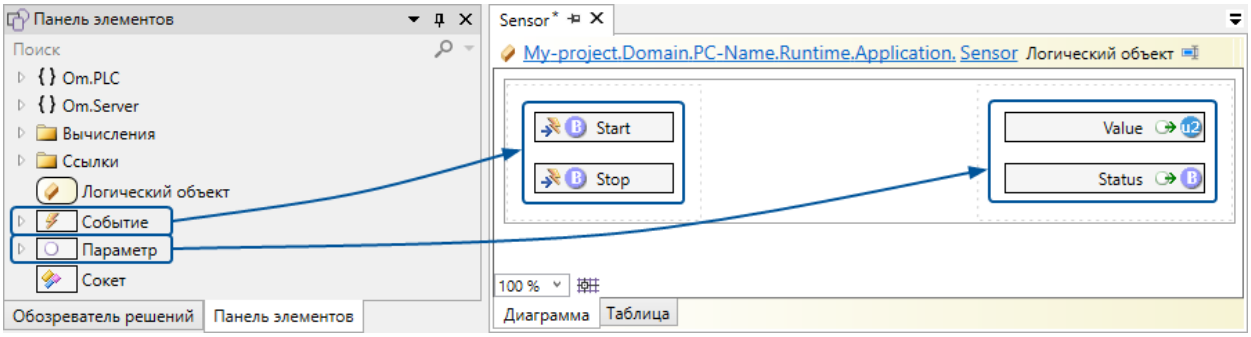



4.2. В **Приложение** добавляем **Логический объект** и указываем ему имя **«Sensor»**.




4.3. В **«Sensor»** добавляем сигналы:

| Имя | Тип элемента | Тип значения | Направление |
|---|---|---|---|
| Start | **Событие** | bool | Вход |
| Stop | **Событие** | bool | Вход |
| Value | **Параметр** | uint2 | Выход |
| Status | **Параметр** | bool | Выход |



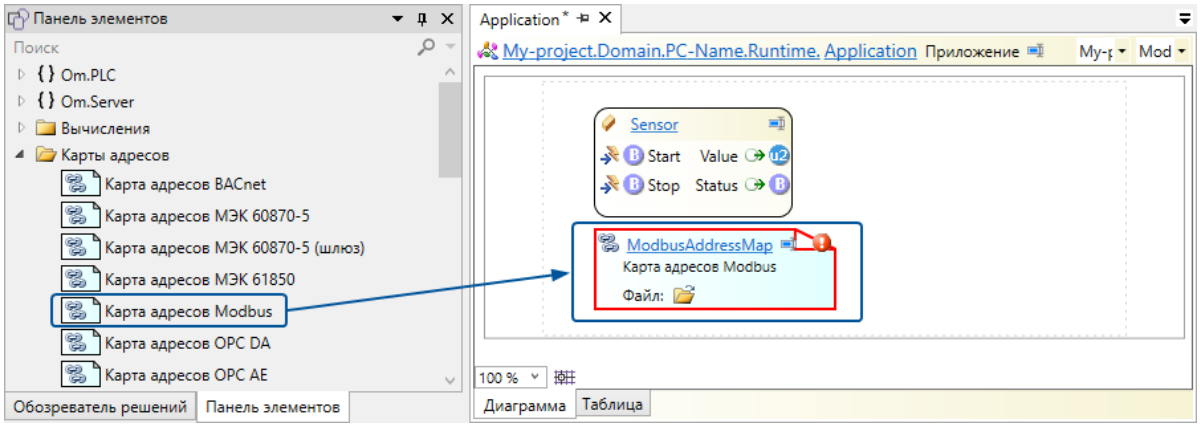

Сигналы **«Value»** и **«Status»** - данные, которые источник будет предоставлять; сигналы **«Stop»** и **«Start»** - команды, которые он будет принимать.

5. Указываем, что исполняющий компонент предоставляет данные по Modbus:

5.1. Для сигналов указываем адреса Modbus:

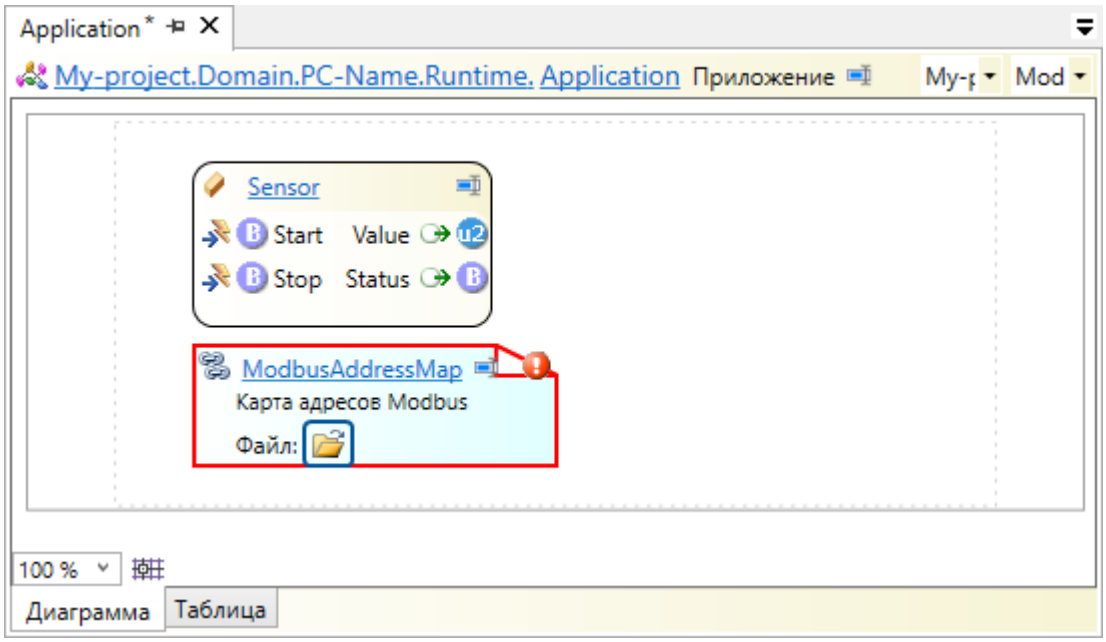
5.1.1. Переходим в **Приложение** (в исполняющем компоненте).

5.1.2. Добавляем элемент **Карта адресов Modbus**.



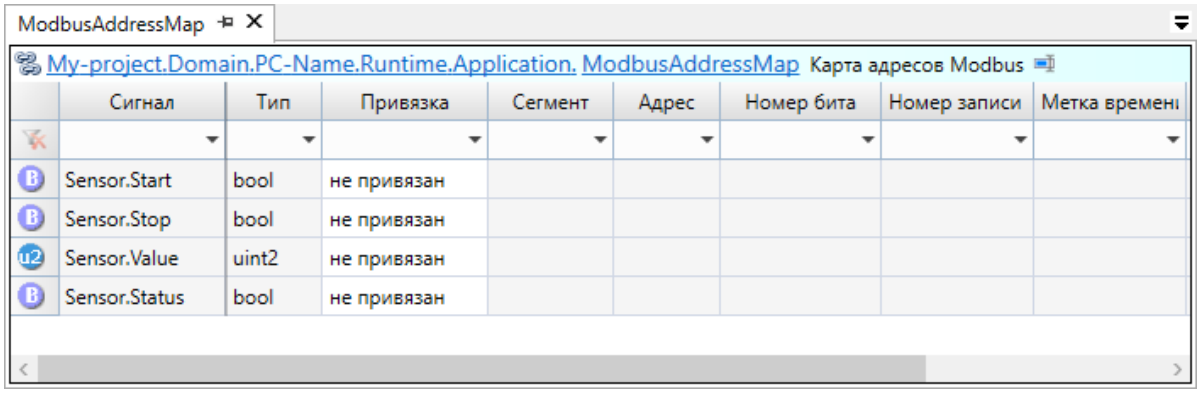

5.1.3. Создаём файл, в котором будут храниться адреса.

Чтобы создать файл, нажмите на иконку папки на изображении карты адресов: откроется окно выбора файла.




5.1.4. Открываем карту адресов.

В карте адресов видим сигналы, которые есть в приложении.



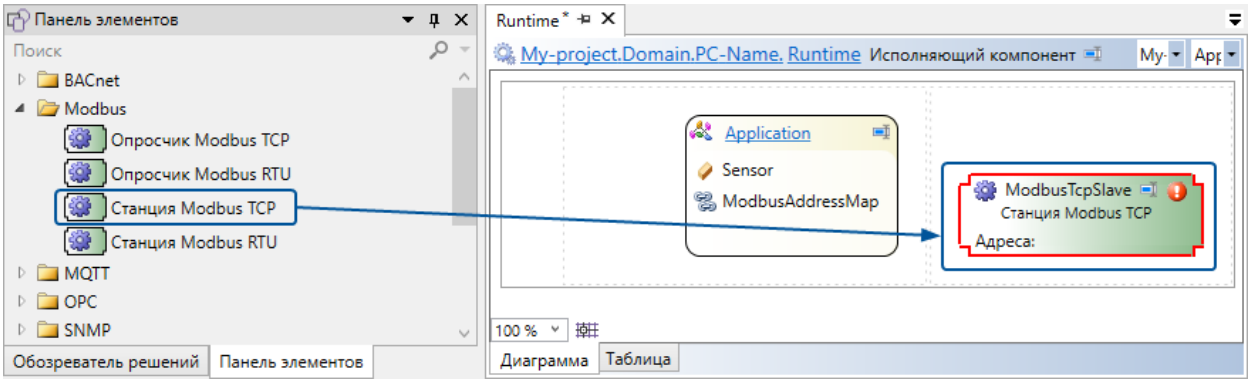

| Сигнал | Тип | Привязка | Сегмент | Адрес | Номер бита | Номер записи | Метка времени |
|---|---|---|---|---|---|---|---|
| Sensor.Start | bool | не привязан | | | | | |
| Sensor.Stop | bool | не привязан | | | | | |
| Sensor.Value | uint2 | не привязан | | | | | |
| Sensor.Status | bool | не привязан | | | | | |

5.1.5. Указываем адреса:

| Сигнал | Привязка | Сегмент | Адрес |
|---|---|---|---|
| Value | непосредственно | Holding Registers | 0 |
| Status | непосредственно | Discretes Input | 0 |
| Start | непосредственно | Coils | 0 |
| Stop | непосредственно | Coils | 1 |

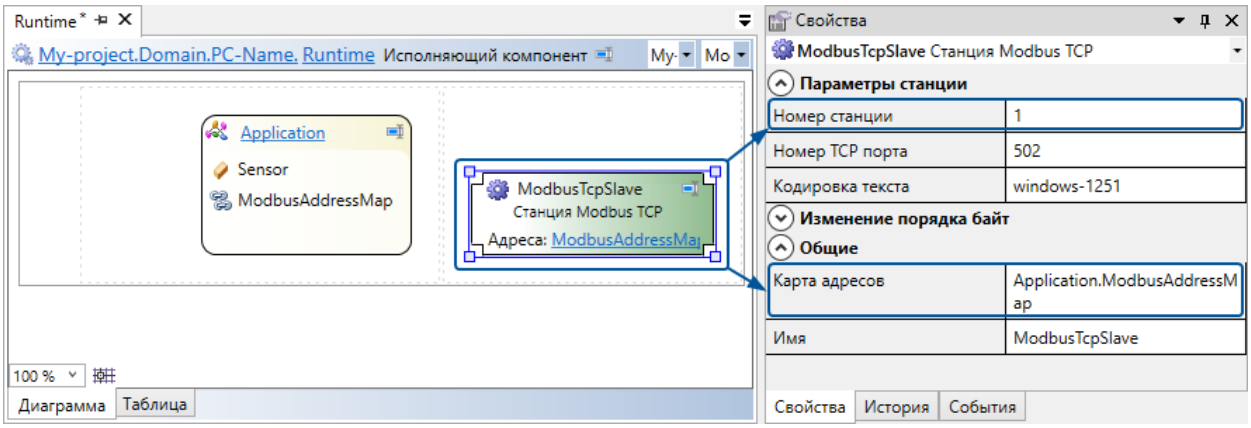
Значения остальных полей менять не нужно.

5.2. В **Исполняющий компонент** добавляем элемент **Станция Modbus TCP**.




5.3. В свойствах станции:

- **Номер станции** - указываем «1».
- **Карта адресов** - выбираем добавленную карту адресов.



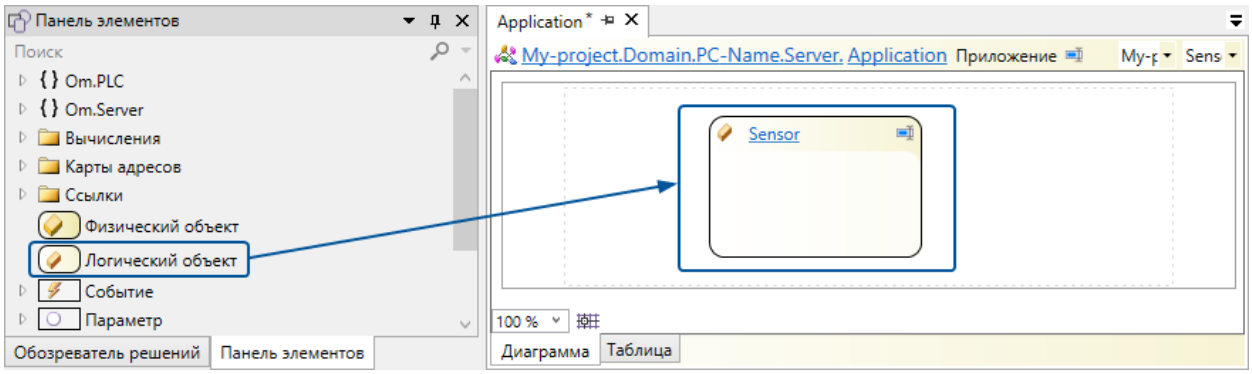

Источник описан. Теперь надо в SePlatform.Data Server настроить сбор данных с источника и передачу ему команд.

## Настраиваем передачу данных

1. Связываем сигналы SePlatform.Data Server с сигналами источника:

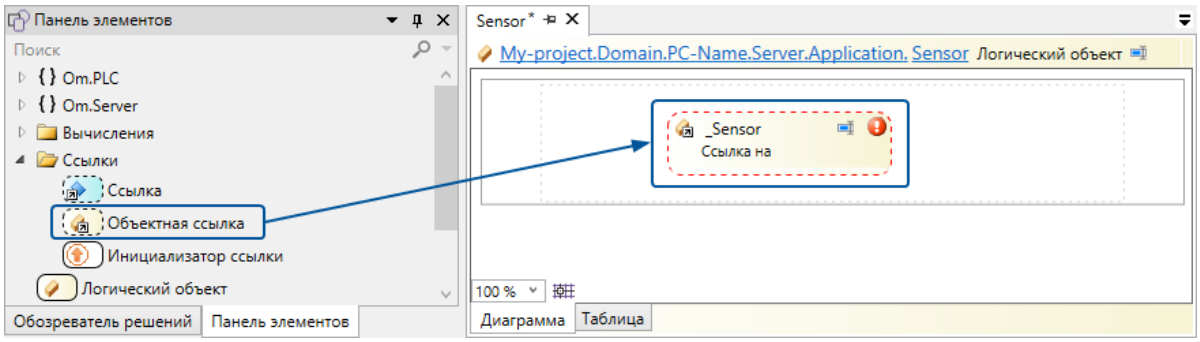
   1.1. Переходим в **Приложение**, размещённое в SePlatform.Data Server. Сигнал, добавленный в предыдущем разделе, можно удалить

1.2. Добавляем **Логический объект**.



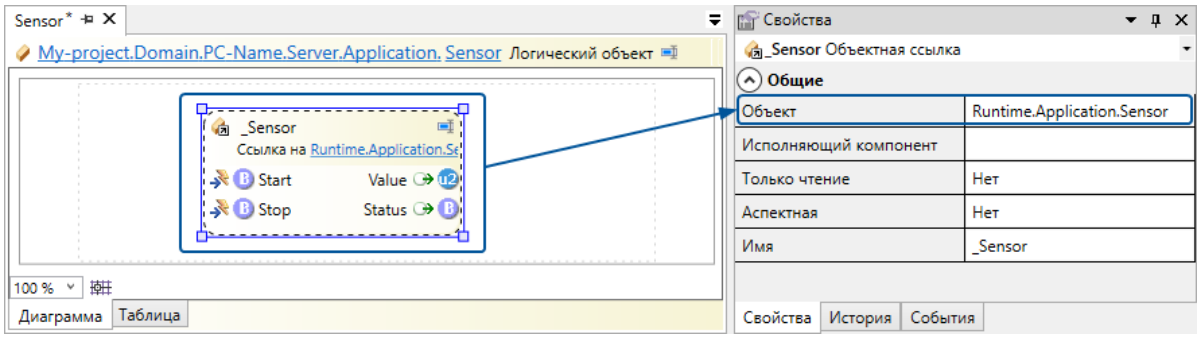

1.3. В него добавляем ссылку на объект «**Sensor**», размещённый в эмуляторе:

1.3.1. Добавляем элемент **Объектная ссылка**.




1.3.2. В свойстве **Объект** выбираем объект, размещённый в эмуляторе.

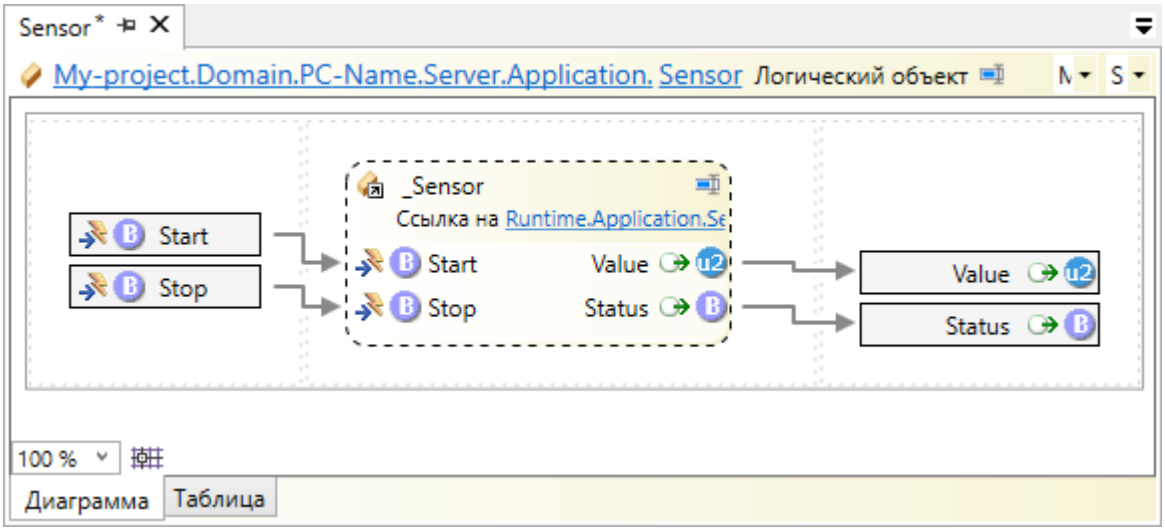
Ссылка примет вид объекта, на который указывает.



1.4. В контекстном меню ссылки выбираем **Экспонировать входы и выходы**.

При экспонировании:

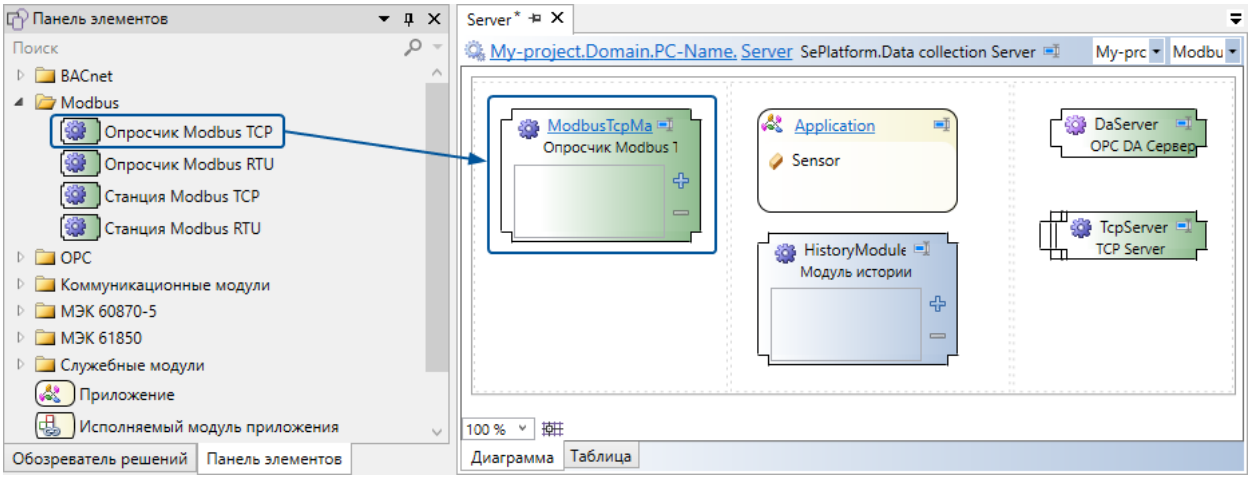
- для каждого входного/выходного сигнала ссылки будет создан аналогичный сигнал в локальном объекте.
- входы и выходы ссылки будут связаны с соответствующими входами и выходами локального объекта.




2. Указываем, что данные передаются по Modbus:

2.1. Переходим в **`SePlatform.Server`**.

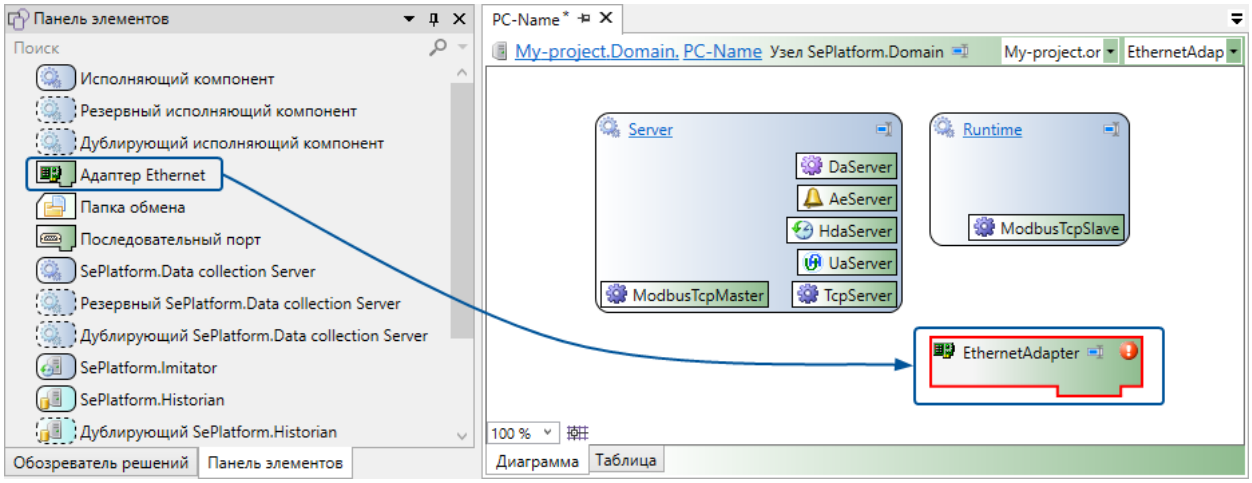
2.2. Добавляем **`Опросчик Modbus TCP`**.



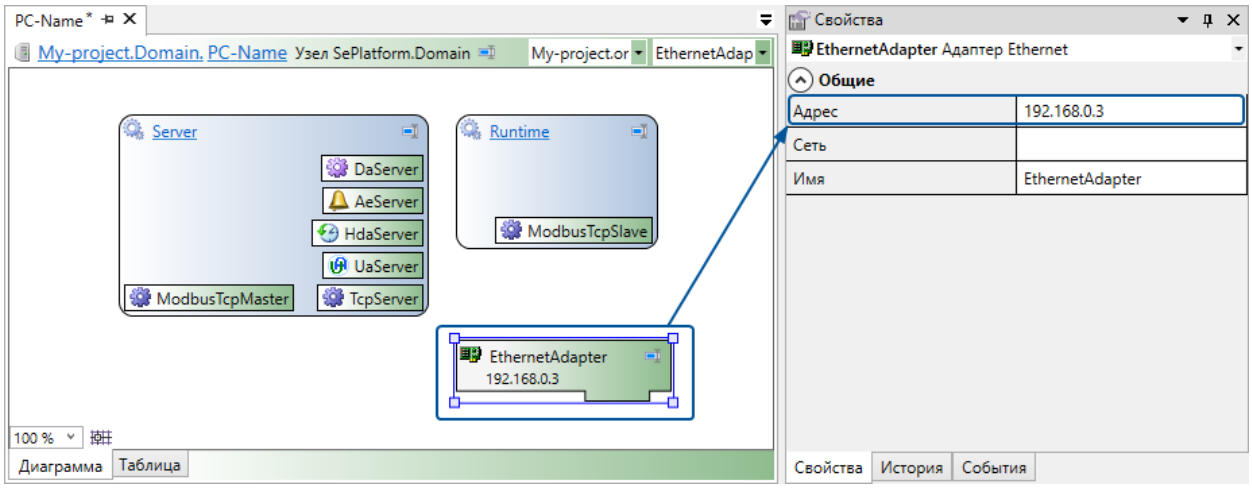

Опросчик будет опрашивать все доступные станции Modbus.

3. Чтобы компоненты могли обмениваться данными, нужно объединить их в сеть:

3.1. В **Узел SePlatform.Domain** добавляем **Адаптер Ethernet**.

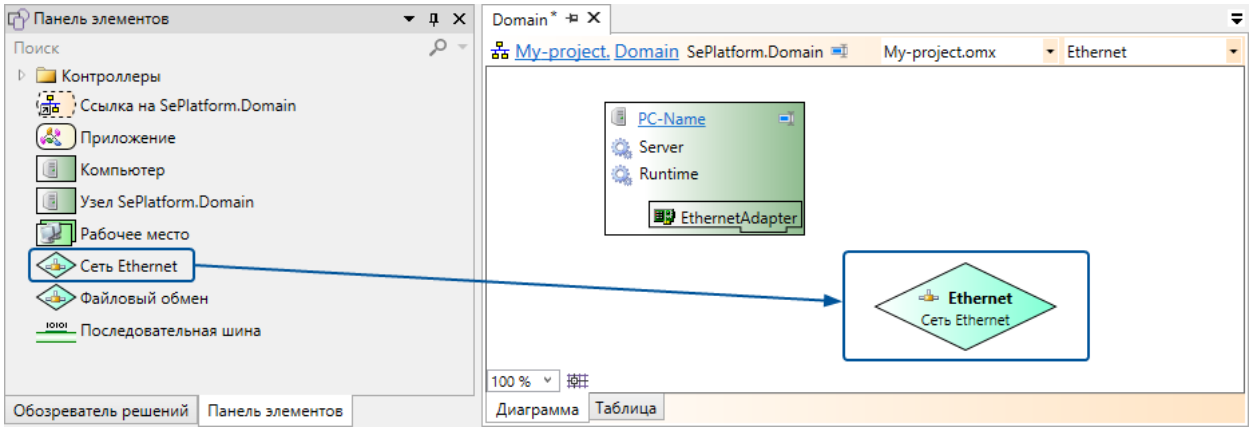



3.2. В свойстве **Адрес** указываем IP-адрес компьютера.




3.3. Если добавляли компьютер, для него делаем то же самое: добавляем **Адаптер Ethernet** и указываем IP-адрес.

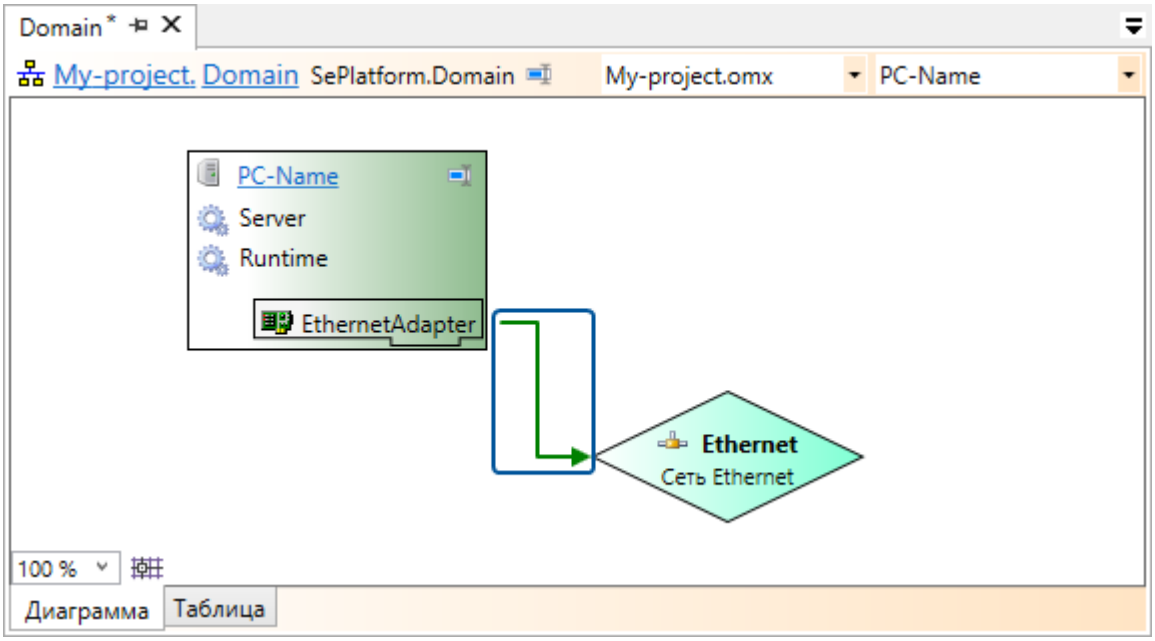
3.4. В **SePlatform.Domain** добавляем **Сеть Ethernet**.



3.5. Соединяем адаптер(ы) с добавленной сетью.

ПРИМЕЧАНИЕ

Чтобы соединить адаптер с сетью, протяните связь от соединительной точки адаптера к любой соединительной точке сети. Соединительные точки появляются при наведении мышью на элемент.



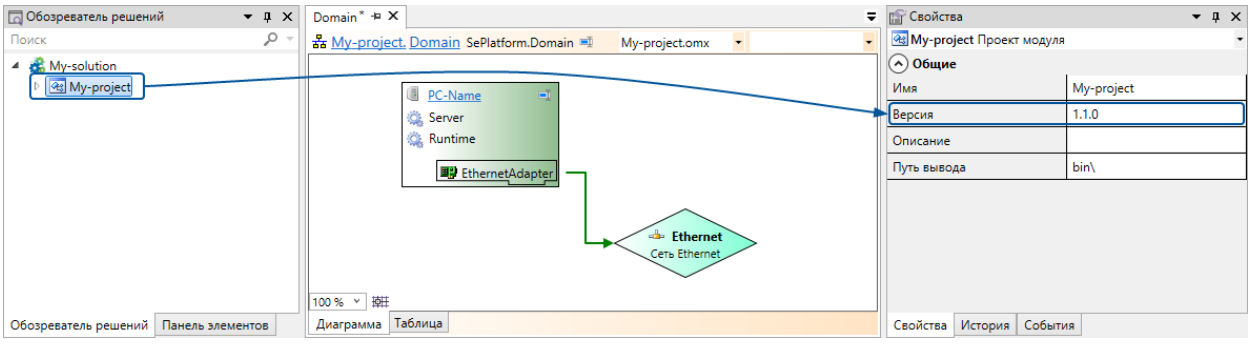

ПРИМЕЧАНИЕ

После соединения у адаптера(ов) в свойстве **Сеть** будет указана сеть, с которой он(и) связаны.

## Проверяем, что данные передаются

1. Выполняем развёртывание:

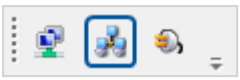
1.1. Меняем версию проекта: в обозревателе решений выбираем проект и в его свойствах указываем новую версию.



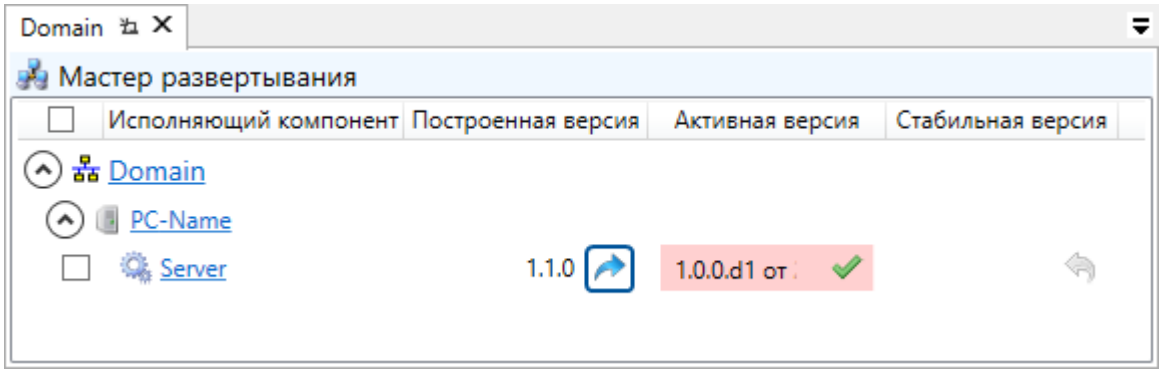

1.2. Строим решение.


Конфигурациям, которые будут созданы при построении, будет назначена новая версия проекта.

1.3. Открываем мастер развёртывания.

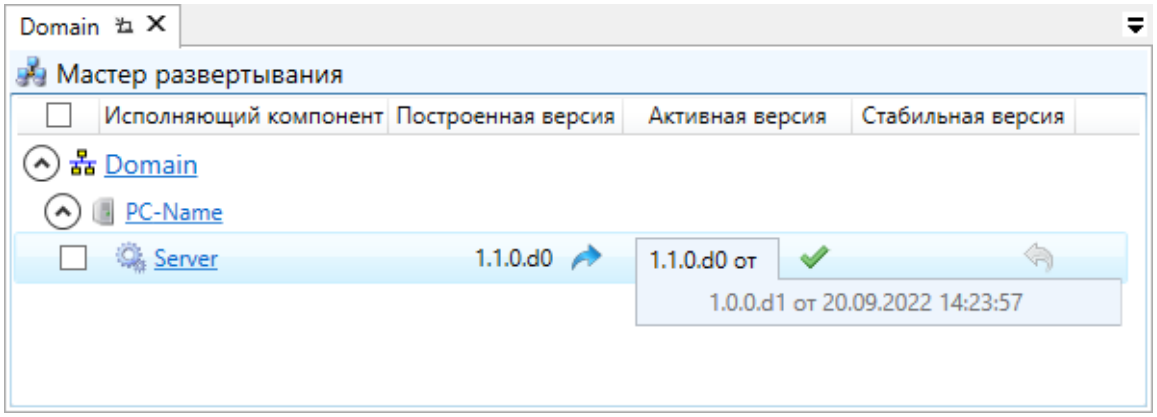


1.4. Применяем конфигурацию к SePlatform.Data Server.



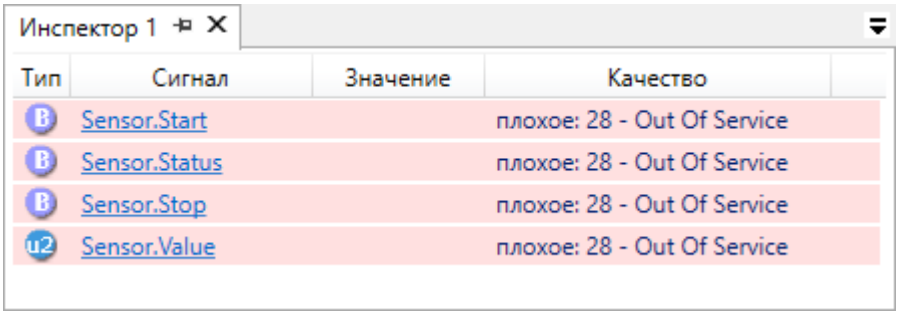

ПРИМЕЧАНИЕ

Теперь в списке версий компонента есть две версии конфигурации. При выборе версии в списке, она будет применена к компоненту. Это позволяет возвращаться к предыдущим построенным конфигурациям.

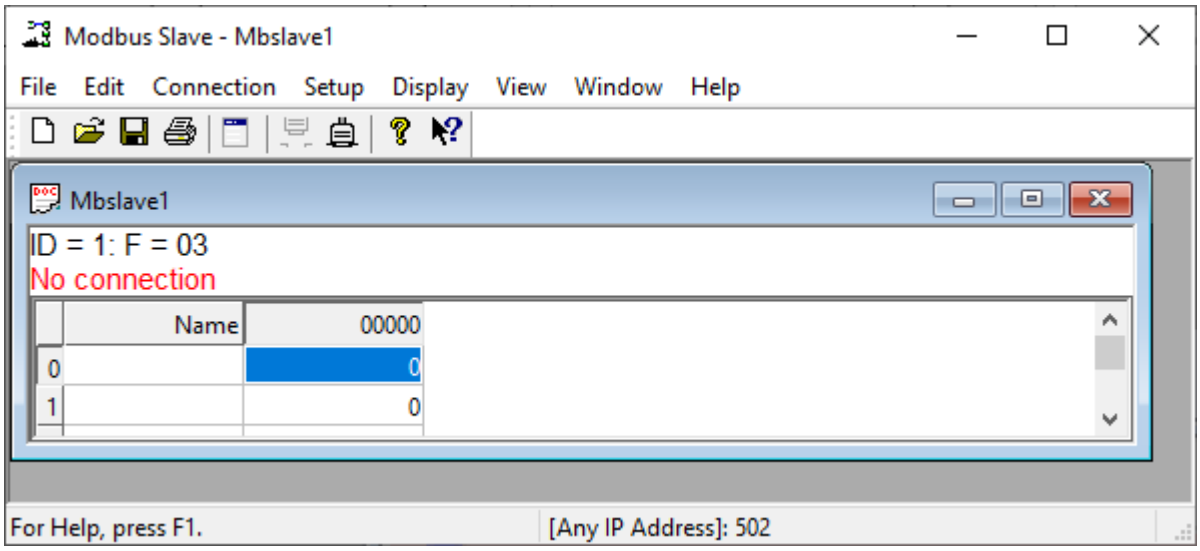



2. Запускаем Service - OPCExplorer.

3. В Service - OPCExplorer в дереве сигналов ищем объект (папку) **«Sensor»** и добавляем все его сигналы в инспектор сигналов.



4. Запускаем эмулятор станции Modbus.

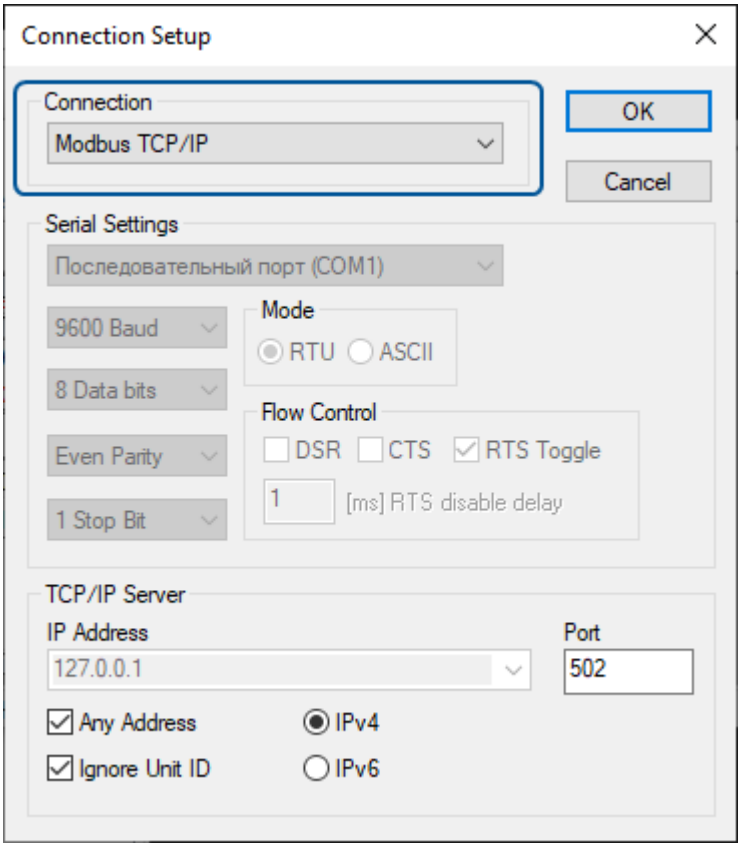



5. Указываем параметры подключения:

   5.1. Меню **`Connection`** → **`Connect...`**.

   Откроется окно настройки подключения.

   5.2. В окне выбираем тип подключения: **`Modbus TCP/IP`**. Остальные параметры оставляем по умолчанию.




   5.3. Нажимаем **`OK`**.

6. Создаём три рабочих поля по числу используемых сегментов: меню **`File`** → **`New`**.

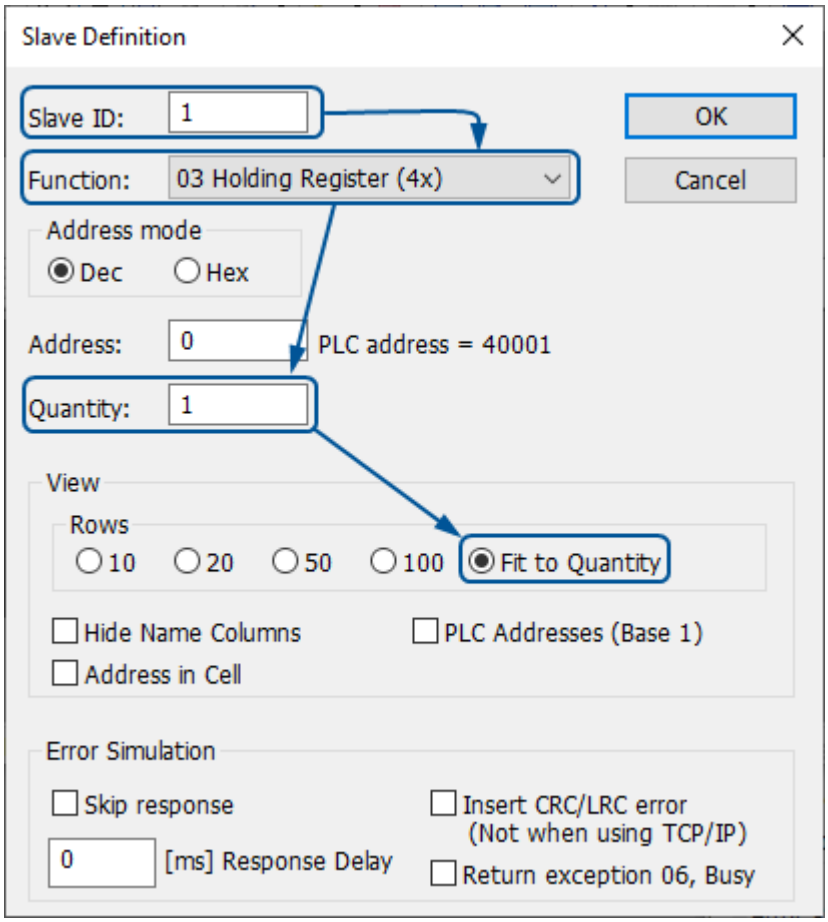
7. Настраиваем каждое рабочее поле:

7.1. В контекстном меню рабочего поля выбираем **`Slave Definition...`**.

Откроется окно настроек.

7.2. В окне указываем:

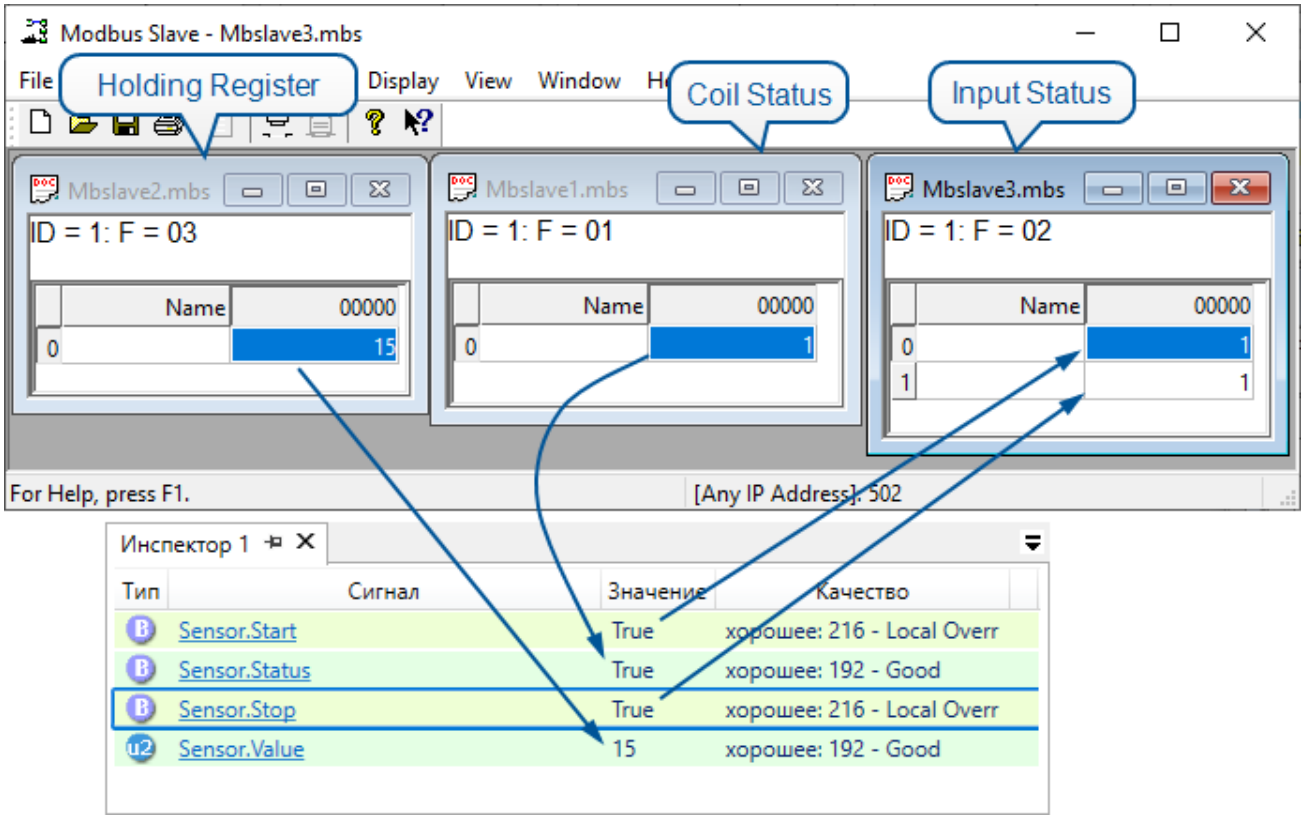
- **`«Slave ID»`** - **`«1»`** (номер станции).
- **`Function`** - в каждом рабочем поле указываем нужный сегмент:
  - **`Holding Register`**
  - **`Coil Status`** (Coils)
  - **`Input Status`** (Discretes Input)
- (Опционально) **`Quantity`** - количество ячеек в сегменте. Указываем столько, сколько сигналов в сегменте: **`«2»`** в **`Coil Status`** и **`«1»`** остальных. Можно оставить по умолчанию: лишние ячейки использовать не будем.
- (Опционально) **`Rows`** - **`«Fit to Quantity»`**: можно выбрать, если указали количество ячеек: в этом случае лишние ячейки отображаться не будут.




7.3. Нажимаем **`OK`**.

8. Проверяем, что значения сигналов передаются между SePlatform.Data Server и станцией:

   - Сегмент **`Holding Register`**: в эмуляторе изменяем значение ячейки с адресом 0 → в Service - OPCExplorer изменяется значение сигнала **«`Value`»** (значение запрашивается из ячейки).
   - Сегмент **`Coil Status`**: в эмуляторе изменяем значение ячейки с адресом 0 → в Service - OPCExplorer изменяется значение сигнала **«`Status`»** (значение запрашивается из ячейки).
   - Сегмент **`Input Status`**: в Service - OPCExplorer изменяем значения сигналов **«`Start`»** и **«`Stop`»** → в эмуляторе изменяются значения ячеек с адресами 0 и 1 соответственно (команды передаются и записываются в ячейки).

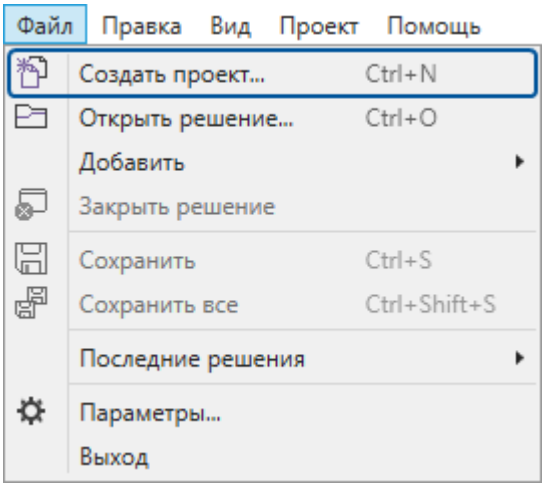


# 4. Решения и проекты

Проект - это то, с чем вы работаете в SePlatform.Development Studio. Проект содержит элементы и файлы, которые вы добавляете. Проект всегда открывается внутри решения, в которое он входит.

Решение - это контейнер для одного или нескольких связанных между собой проектов. Само по себе решение не содержит данных; работа с решением - это работа с входящими в него проектами.

Работа в SePlatform.Development Studio начинается с создания проекта. Чтобы создать проект, выберите в меню **Файл** → **Создать проект...**: откроется мастер создания проекта.




При создании проекта, создаётся также содержащее его решение. Если в SePlatform.Development Studio уже открыто решение, при создании проекта решение создаваться не будет. Вместо этого созданный проект будет добавлен в открытое решение.

После создания проекта, можно приступать к описанию его содержимого: см. главу 5. Разработка проекта (стр. 36).

# 5. Разработка проекта

ПРИМЕЧАНИЕ

Пример создания проекта приведён в разделе 3.1. Создание простого проекта (стр. 13).

При разработке проекта в нём описываются домен и объекты, которые исполняются в этом домене.

Домен - это среда, в которой будет исполняться проект. Описание домена содержит информацию о том, какие компоненты есть в домене, где они расположены и как связываются друг с другом. В описании домена указываются данные реальной среды исполнения включая сетевые имена и IP-адреса компьютеров, а также интерфейсы и протоколы, по которым компоненты передают данные. В рамках одного проекта может быть описано несколько сред исполнения. Тогда при применении изменений проекта необходимо выбрать среду, к которой должны быть применены изменения.

Объекты - это кирпичики данных. Они размещаются в компонентах домена и описывают данные, с которыми этот компонент работает: передаёт или обрабатывает. Для каждого объекта описываются его данные, а также связи с другими объектами: как внутри того же компонента, так и с объектами, размещёнными в других компонентах. Описание объектов не зависит от среды, в которой они исполняются: при изменении расположения компонентов домена или протоколов передачи данных, изменять объекты не нужно. Объекты можно описывать как непосредственно внутри описания домена, так и с помощью типов, которые описываются отдельно.

При разработке проекта не существует единственно правильной последовательности шагов. Вы можете сначала описать домен, а после описывать в нём объекты. Или наоборот, сначала описать типы объектов, а после этого описать домен и разместить в нём объекты описанных типов. При работе с уже существующим решением, вам потребуется изменять как описание домена (добавлять новые компоненты или редактировать старые), так и описание объектов (добавлять или редактировать сигналы в существующих объектах, добавлять новые объекты).

Основные этапы разработки проекта приведены в первых двух разделах данной главы:

- Описание домена (стр. 36) - в разделе описано, как создать домен, добавить в него компоненты, объединить их в сети, а также как создать приложения - элементы, в которых размещаются объекты, исполняемые в каждом компоненте.
- Добавление объектов (стр. 56) - в разделе описано добавление объектов вне зависимости от того, описываются ли они в домене или с помощью типов. В разделе описаны: добавление сигналов, добавление связей с другими объектами, описание вычислений и генерации событий, а также как разместить объекты, если они описаны с помощью типов.

В остальных разделах описано решение задач, которые могут возникнуть в процессе разработки проекта, а также возможности SePlatform.Development Studio, которые можно применять при разработке проектов.

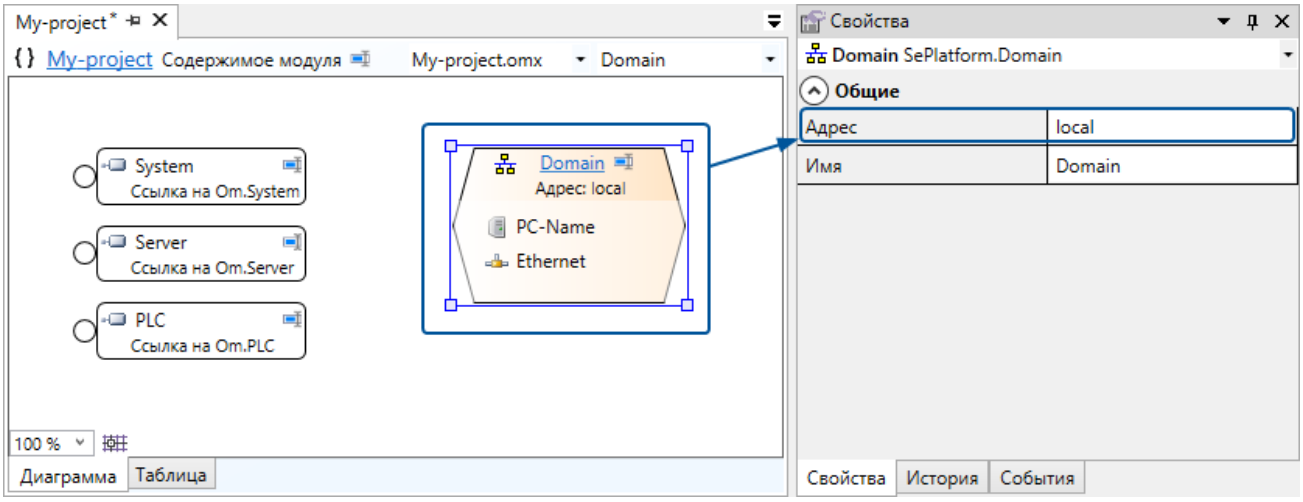
После того, как будет описан домен и исполняемые в нём объекты, можно выполнять развёртывание (стр. 154).

## 5.1. Описание домена

1. Выберите среду исполнения и настройте связь с ней (стр. 37).
2. Перейдите в **Содержимое модуля** (корень проекта) или добавьте **Пространство имён** и перейдите в него.

3. Добавьте элемент **SePlatform.Domain**. Элемент по умолчанию есть в проекте, если он создан из шаблона **Проект развёртывания**.

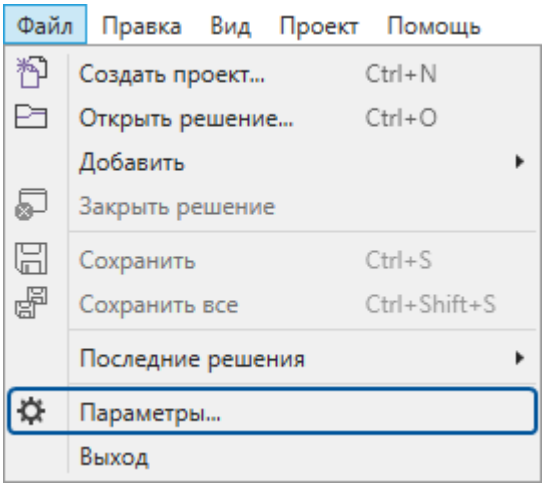
В свойстве **Адрес** укажите имя центрального узла домена.




4. Добавьте в **SePlatform.Domain** компоненты домена (стр. 39).
5. Для компонентов домена добавьте приложения и разместите в них объекты (стр. 44).
6. Настройте передачу данных (стр. 48).

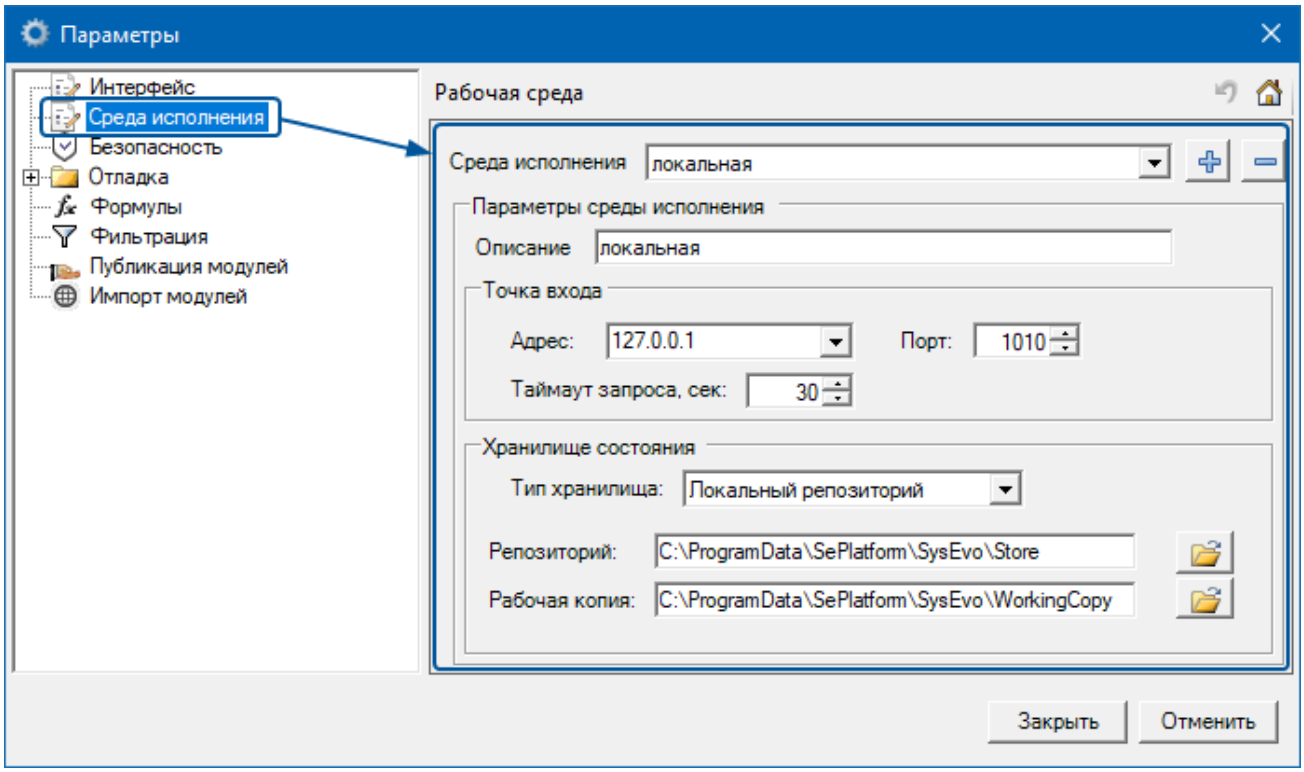
## 5.1.1. Настройка связи со средой исполнения

1. Откройте окно параметров: в меню **Файл** → **Параметры...**.




Откроется окно параметров приложения.

2. В узле **Среда исполнения** добавьте новую среду исполнения или выберите существующую из списка и укажите её параметры.




Параметры среды исполнения:

2.1. В поле **Описание** укажите любое имя.

2.2. В области **Точка входа** укажите параметры подключения к любому компьютеру домена, на котором установлен SePlatform.Net.Agent (входит в дистрибутив SePlatform.Domain):

- **Адрес** - IP-адрес компьютера, на котором функционирует SePlatform.Net.Agent;
- **Порт** - значение атрибута **NetEnterPort** в конфигурации SePlatform.Net.Agent;
- **Таймаут запроса, сек** - время ожидания ответа между попытками подключения.

2.3. В области **Хранилище состояния** выберите тип хранилища:

- **«Локальный репозиторий»** - папка на локальном компьютере.
- **«Удаленный репозиторий»** - папка на сервере SVN.

При выборе типа хранилища отобразятся поля параметров для хранилищ этого типа.

- Параметры локального репозитория:

| Параметр | Описание |
|---|---|
| **Репозиторий** | Папка основного хранилища идентификаторов элементов. |
| **Рабочая копия** | Папка для хранения идентификаторов, находящаяся под контролем версий SVN. Рабочая копия предназначена для сохранения идентификаторов элементов при разработке конфигураций. После построении решения, идентификаторы элементов сохраняются в репозиторий. |

- Параметры удалённого репозитория:

| Параметр | Описание |
|---|---|
| **Репозиторий** | Наименование основного хранилища идентификаторов элементов. |
| **Пользователь** | Пользователь SVN. |
| **Пароль** | Пароль пользователя SVN. |

О том, как работать с удалённым репозиторием, читайте в разделе 5.3. Совместная разработка проекта несколькими пользователями (стр. 89).

3. Нажмите кнопку **Закрыть** - изменения будут сохранены.

Добавленная среда исполнения появится в выпадающем списке в панели инструментов. Переключение между средами исполнения используется при выборе домена, в котором выполняется развёртывание.


## 5.1.2. Добавление компонентов домена

В домен добавляются компоненты, для которых SePlatform.Development Studio создаёт конфигурации:

- SePlatform.Data Server
- SePlatform.AccessPoint

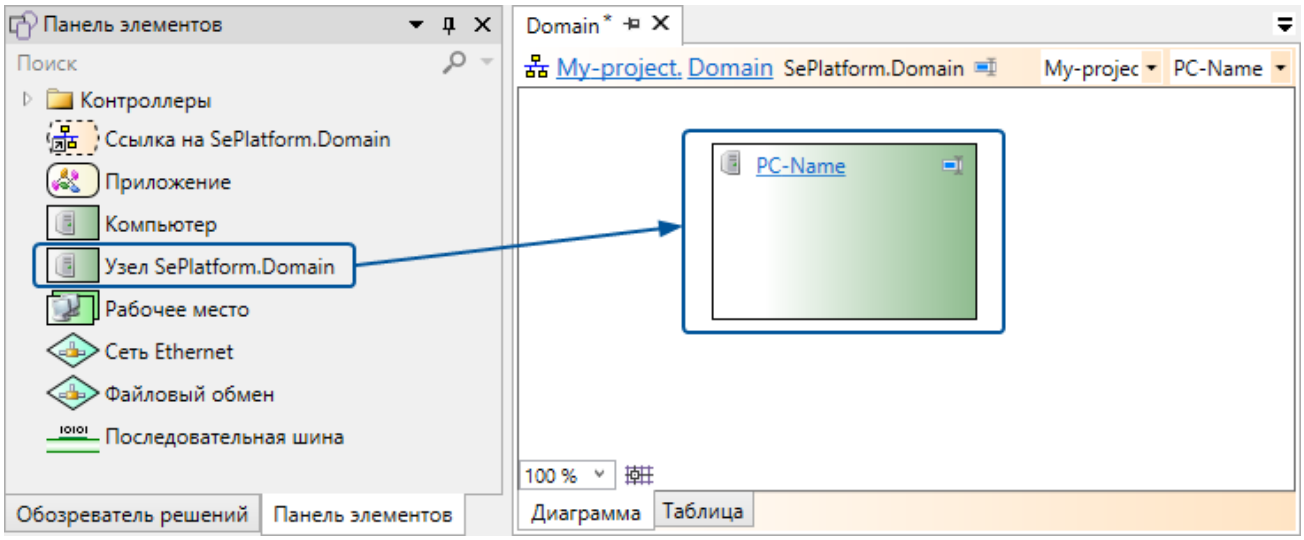
Кроме них в домен добавляются компоненты и устройства, с которыми они взаимодействуют (получают и передают данные):

- серверы истории
- контроллеры
- исполняющие компоненты

## 5.1.2.1. Добавление SePlatform.Data Server

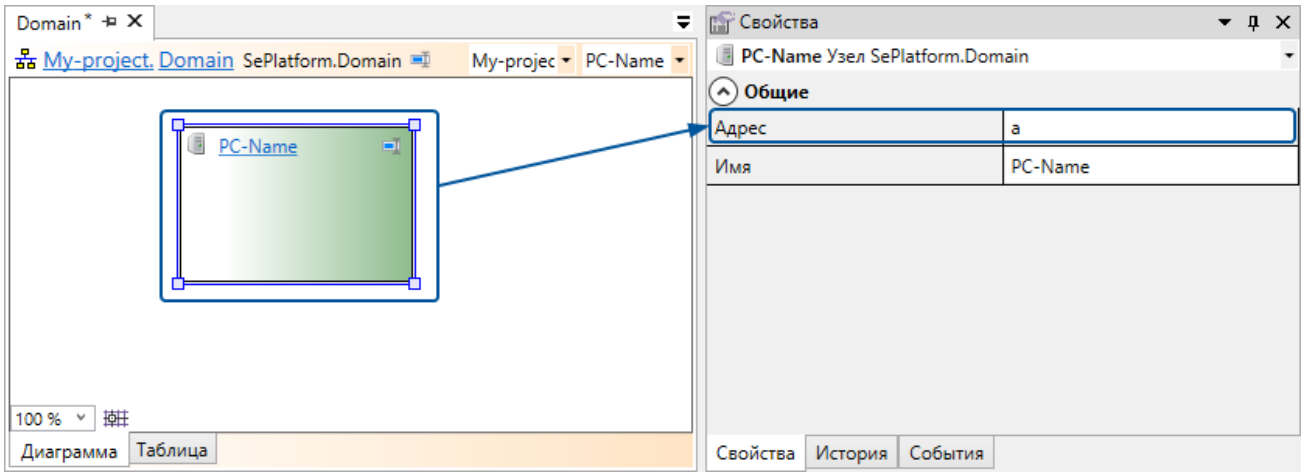
1. В элемент **SePlatform.Domain** добавьте **Узел SePlatform.Domain** - это компьютер, на котором

расположен SePlatform.Data Server.




В качестве имени элемента укажите сетевое имя компьютера.

Если узел не является центральным, в свойстве **Адрес** укажите его имя (такое же, как настроено в SePlatform.Domain). Для центрального узла адрес указывать не нужно: его адрес указывается в свойствах элемента **SePlatform.Domain**.

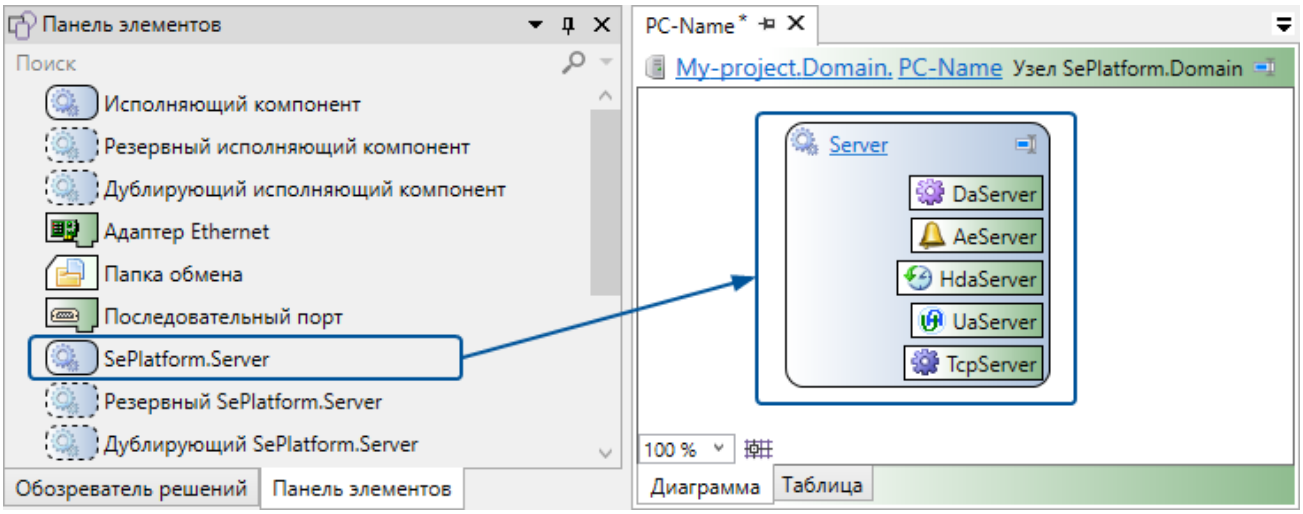



ПРИМЕЧАНИЕ

Если нужный компьютер уже описан с помощью элемента **Компьютер** (компьютер, не имеющий компонентов Систэм Платформ), не удаляйте его. Вместо этого измените тип элемента на **Узел SePlatform.Domain** (выполняется в контекстном меню элемента).

2. Перейдите в добавленный элемент.

3. Добавьте **SePlatform.Server**.




Имя элемента должно совпадать с именем, настроенным в SePlatform.Domain.

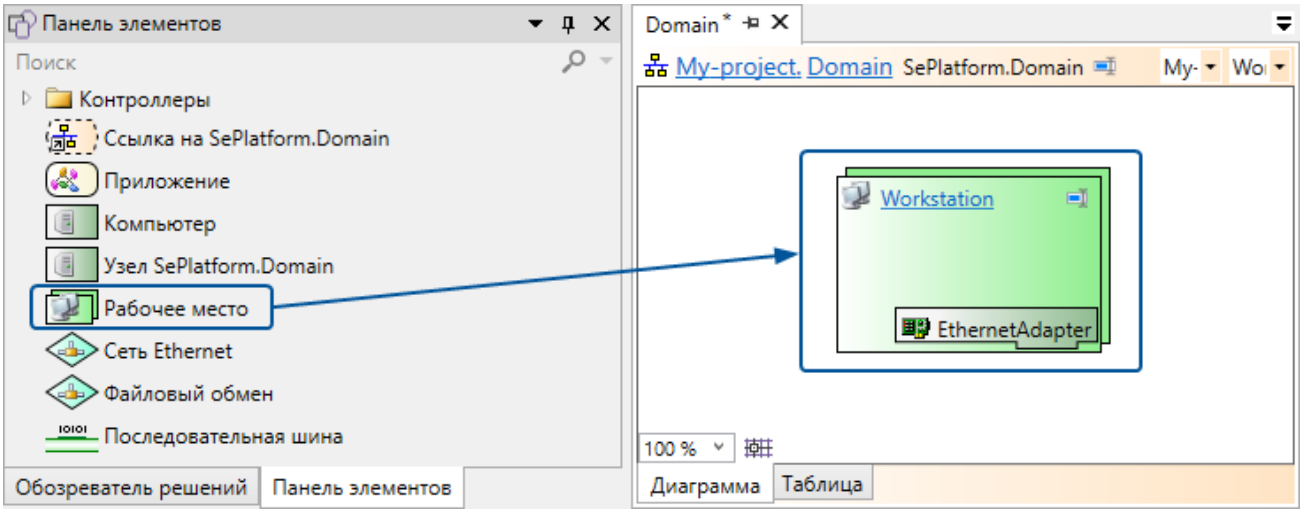
После добавления SePlatform.Data Server, добавьте исполняемое в нём приложение (стр. 44).

ПРИМЕЧАНИЕ

Для повышения надёжности, в проект можно добавить резервный SePlatform.Data Server. Добавление резервного SePlatform.Data Server описано в разделе 5.15. Повышение надёжности проекта автоматизации (стр. 135).

## 5.1.2.2. Добавление SePlatform.AccessPoint

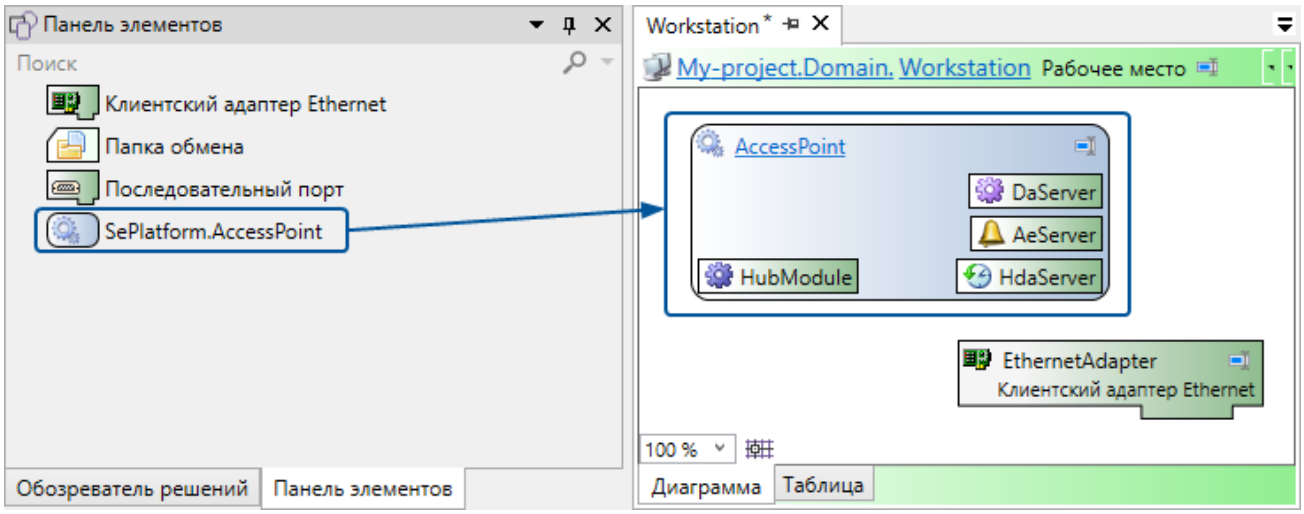
1. В элемент **SePlatform.Domain** добавьте **Рабочее место** - это АРМ оператора.




В отличие от других сетевых устройств, описываемых в домене, одному элементу **Рабочее место** в среде исполнения может соответствовать любое количество АРМов, имеющих одинаковое описание и роль в домене. В качестве имени укажите название роли АРМов. Оно должно совпадать с названием роли, настроенным в SePlatform.Domain.

2. Перейдите в добавленный элемент.

3. Добавьте элемент **SePlatform.AccessPoint**.




Имя элемента должно совпадать с именем, настроенным в SePlatform.Domain.

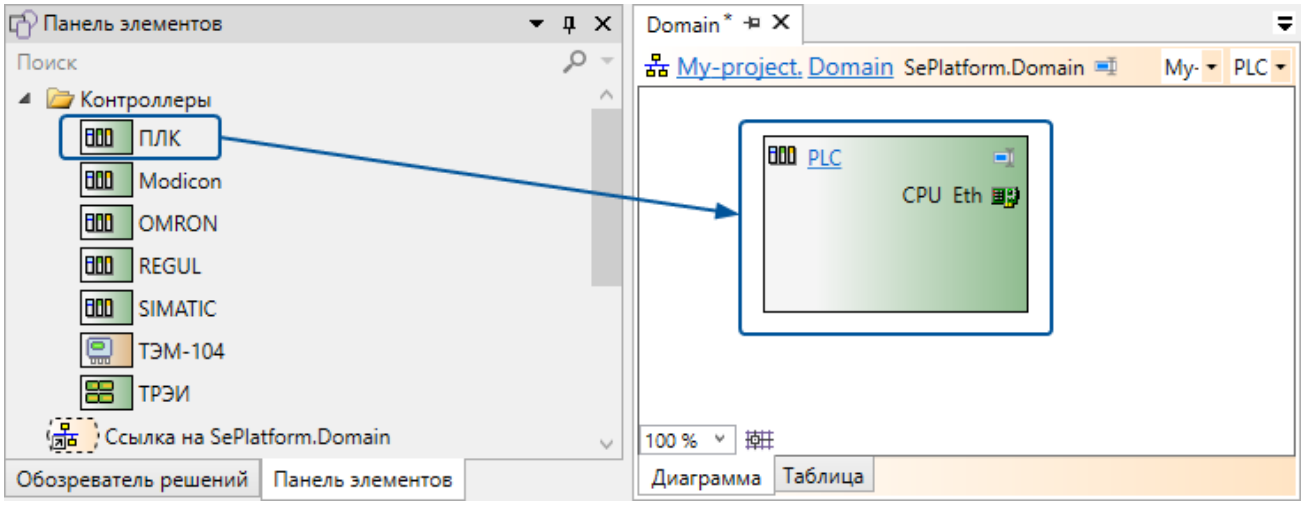
После добавления SePlatform.AccessPoint, добавьте исполняемое в нём приложение (стр. 44).

## 5.1.2.3. Добавление серверов истории

Добавление серверов истории описано в разделе 5.4. Сохранение значений и событий (стр. 94).

## 5.1.2.4. Добавление контроллеров

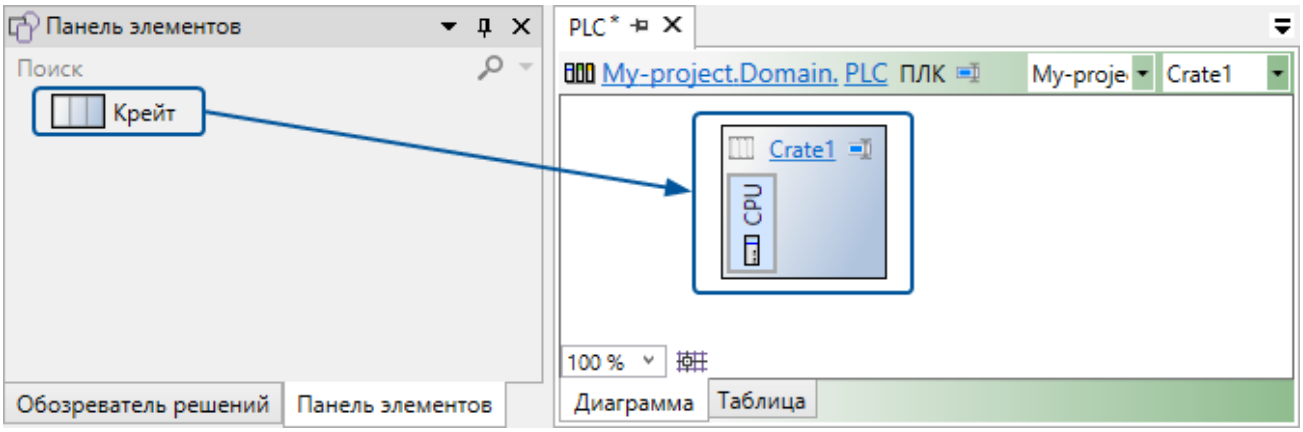
1. В элемент **SePlatform.Domain** добавьте **ПЛК**.




В качестве имени элемента укажите сетевое имя контроллера.

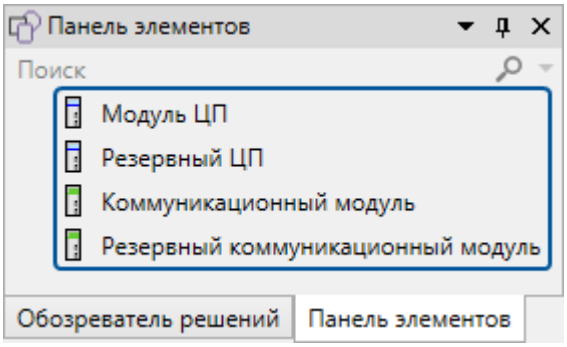
2. Перейдите в добавленный элемент.

3. Добавьте **Крейт** - панель, в которой описываются модули контроллера.




4. Перейдите в добавленный элемент.
5. Добавьте модули, описывающие устройство контроллера:

| Модуль | Описание |
|---|---|
| **Модуль ЦП** | Обязательный модуль. Контроллер может не иметь других модулей кроме него. |
| **Резервный ЦП** | Резервирует основной **Модуль ЦП**. После добавления нужно в свойствах указать **Модуль ЦП**, который он резервирует. |
| **Коммуникационный модуль** | Обеспечивает связь с другими сетевыми устройствами. |
| **Резервный коммуникационный модуль** | Резервирует **Коммуникационный модуль**. После добавления нужно в свойствах указать **Коммуникационный модуль**, который он резервирует. |




После добавления ПЛК, добавьте исполняемое в нём приложение (стр. 44).

## 5.1.2.5. Добавление исполняющих компонентов

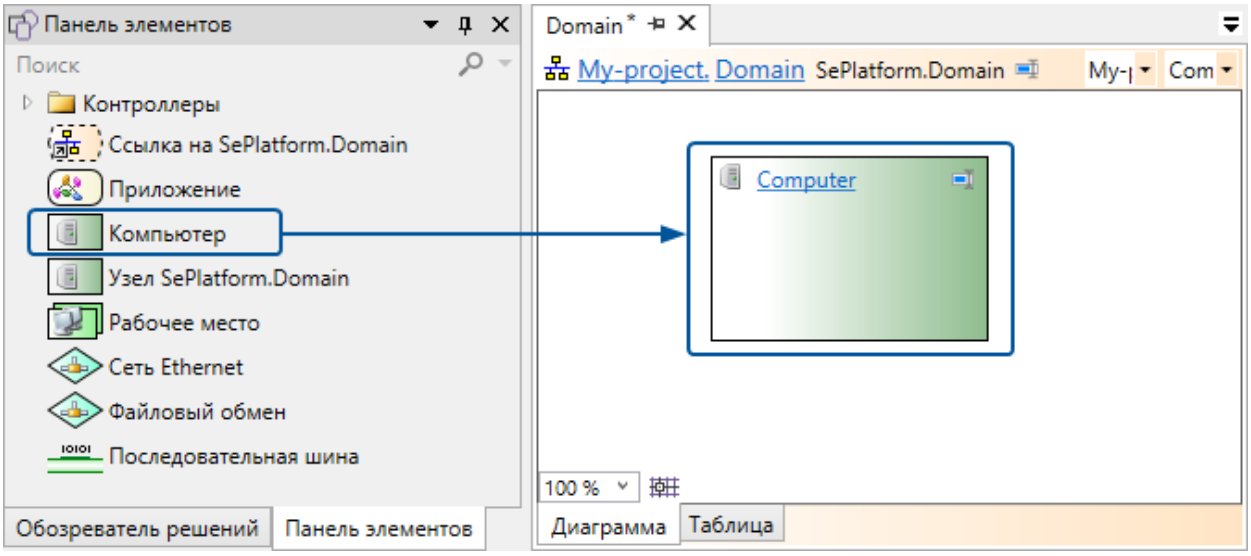
Исполняющий компонент - это информационная система, с которой взаимодействуют компоненты Систэм Платформ. С помощью исполняющего компонента описываются как программы (например, службы), так и сетевые устройства (маршрутизаторы, датчики и другие).

Чтобы добавить исполняющий компонент:

1. Если исполняющий компонент расположен на компьютере, который уже описан (элементы **Узел SePlatform.Domain** и **Компьютер**), перейдите в него.

Если компьютер или сетевой устройство ещё не описано:

1.1. В элемент **SePlatform.Domain** добавьте **Компьютер**.

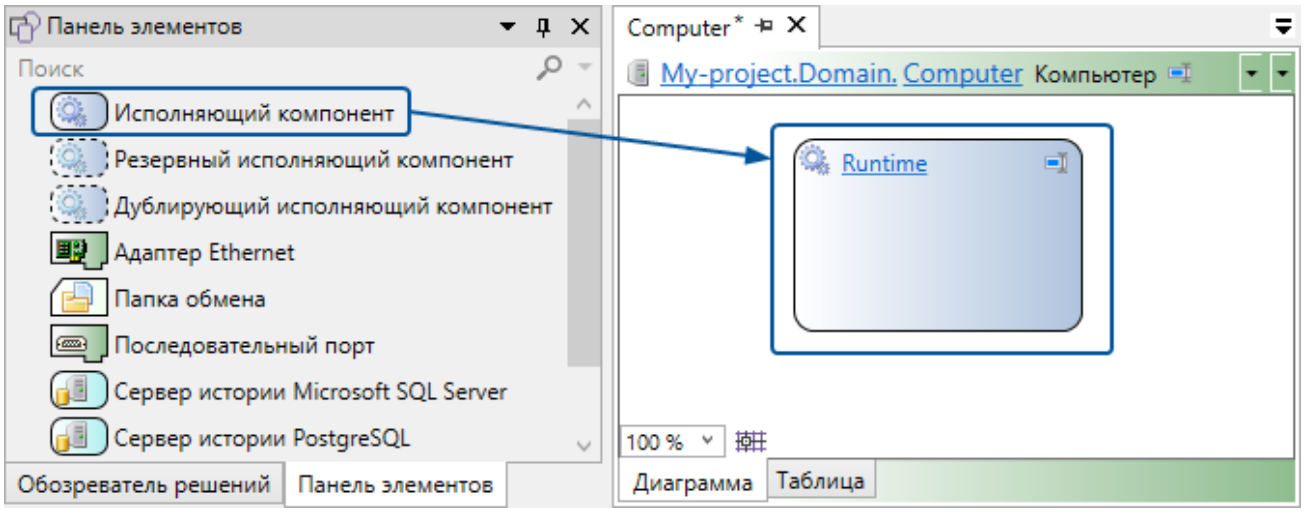



Элемент **Компьютер** описывает компьютер или иное сетевое устройство.

1.2. В качестве имени элемента укажите сетевое имя компьютера/устройства.

1.3. Перейдите в добавленный элемент.

2. Добавьте **Исполняющий компонент**.




В исполняющем компоненте описываются данные компонента и интерфейсы, по которым эти данные предоставляются/запрашиваются.

После добавления исполняющего компонента, добавьте исполняемое в нём приложение (стр. 44).

ПРИМЕЧАНИЕ

Если исполняющий компонент резервируется, в домен нужно добавить резервный исполняющий компонент. Добавление резервного исполняющего компонента описано в разделе 5.15. Повышение надёжности проекта автоматизации (стр. 135).

## 5.1.3. Добавление приложений

Приложения используются для того, чтобы разместить объекты в компонентах домена.

Чтобы добавить приложение:

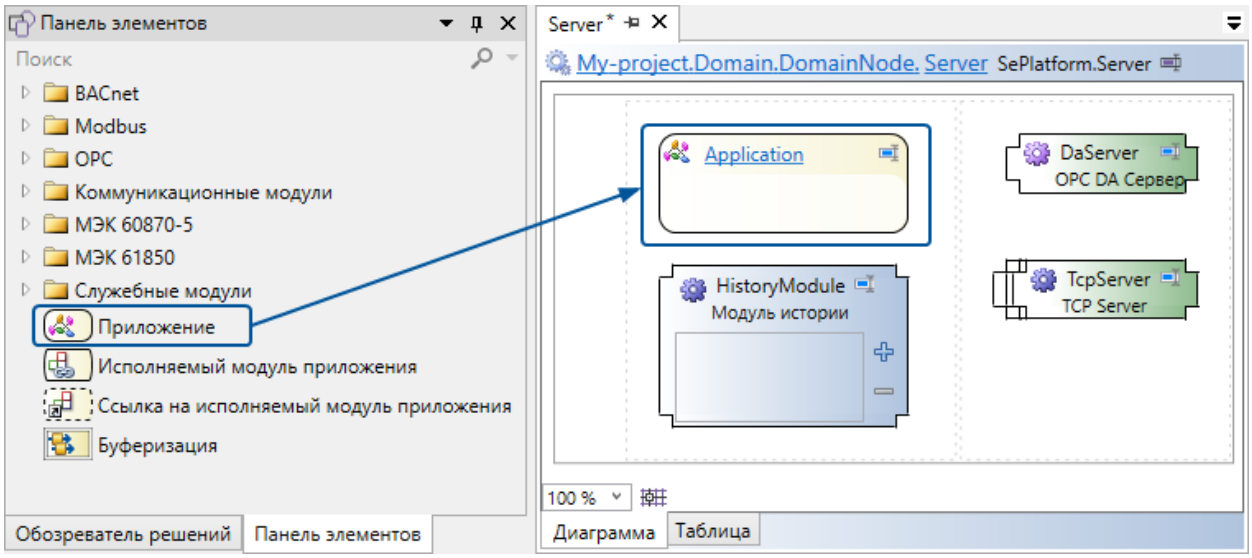
1. Создайте приложение.

   Приложение можно создать в компоненте или в элементе **SePlatform.Domain**. Первый способ самый простой: приложение расположено там, где исполняется. Второй способ более гибкий: приложение находится вне компонента, в котором исполняется, поэтому ссылки на объекты приложения не ломаются при изменениях пути к компоненту (например, при переносе компонента на другой компьютер).

   Чтобы создать приложение в компоненте:

   1.1. Перейдите в компонент.

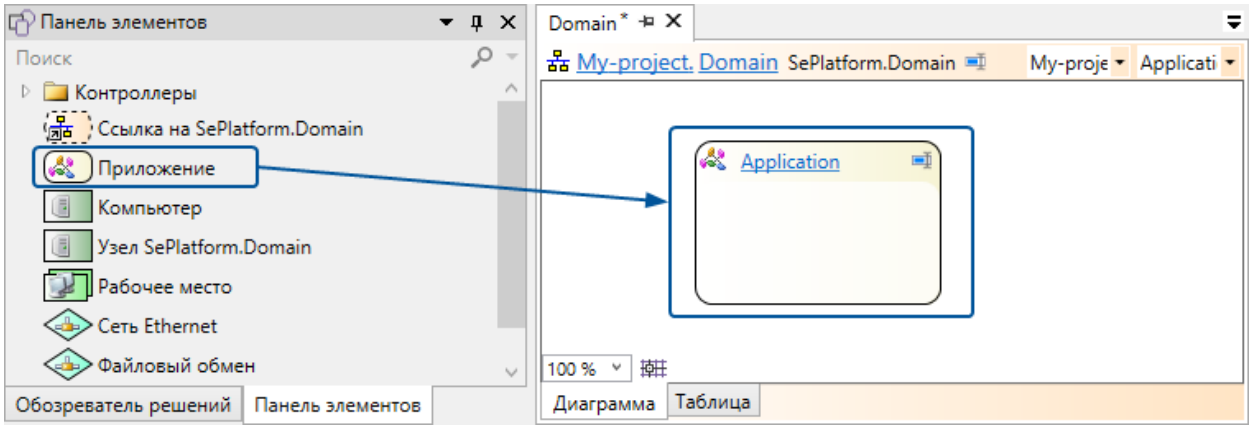
   1.2. Добавьте элемент **Приложение**.




   Чтобы создать приложение в **SePlatform.Domain**:

   1.1. Перейдите в **SePlatform.Domain**.

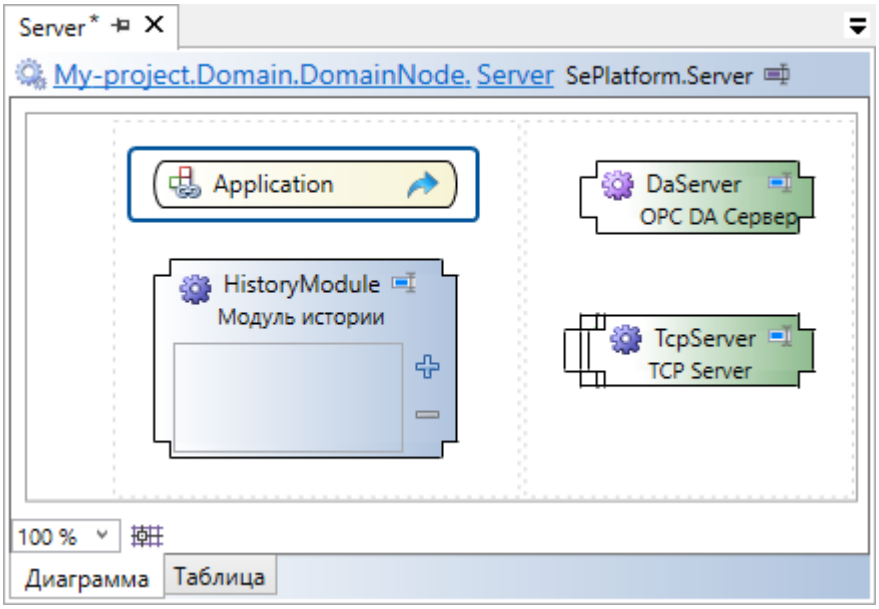
   1.2. Добавьте элемент **Приложение**.




   1.3. Перейдите в компонент, в котором оно должно исполняться.

1.4. В контекстном меню выберите **Разместить исполняемые объекты...** и в появившемся окне выберите приложение.

В компонент будет добавлен **Исполняемый модуль приложения** - ссылка на выбранное приложение.



2. Добавьте в **Приложение** объекты.

Как добавлять объекты описано в разделе 5.2. Добавление объектов (стр. 56).

Объекты можно добавлять в элемент **Приложение** или создать тип приложения и добавлять объекты в него, а приложению указать его тип. Тип приложения позволяет:

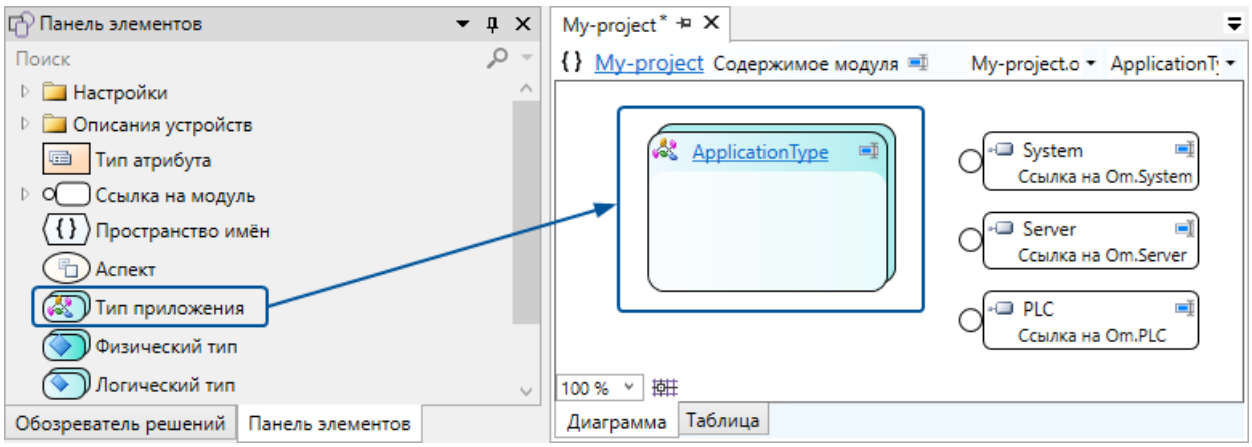
- описывать объекты приложения независимо от домена, в котором размещается само приложение.
- использовать один и тот же тип для нескольких приложений (например при описании одинаковых датчиков).
- использовать один и тот же тип для приложений, размещённых в разных доменах (например для создания тестового домена, в котором будет проверяться работа приложений).

Чтобы использовать тип приложения:

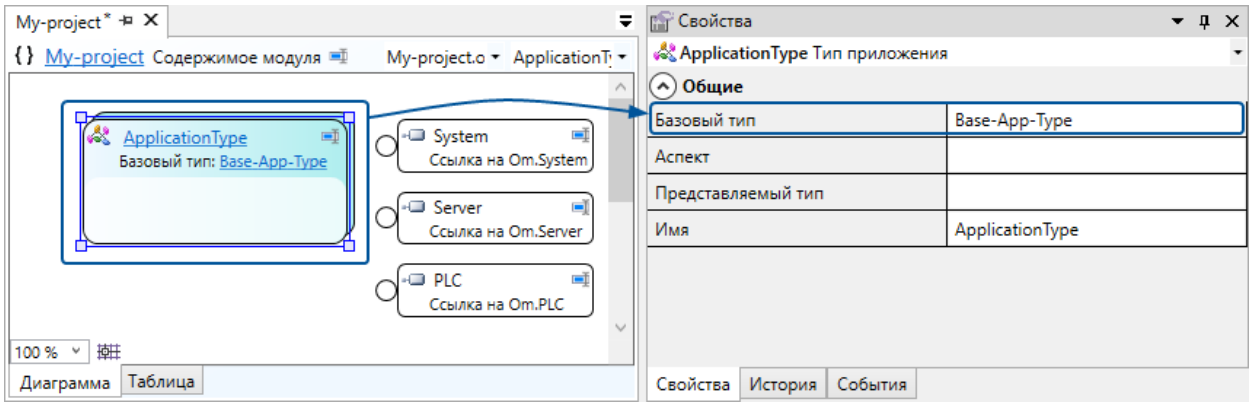
2.1. Перейдите в **Содержимое модуля** (корень проекта) или добавьте **Пространство имён** и перейдите в него.

> ПРИМЕЧАНИЕ
>
> Тип приложения можно добавлять также внутрь других элементов: типов приложений и типов, однако мы рекомендуем этого не делать, чтобы не создавать путаницы. Визуально будет казаться, что в элементе размещён элемент с данными, хотя на самом деле тип приложения - это только описание, поэтому в элементе этих данных не будет.

2.2. Добавьте **Тип приложения**. Элемент по умолчанию есть в проекте, если он создан из шаблона **Проект приложения...**.

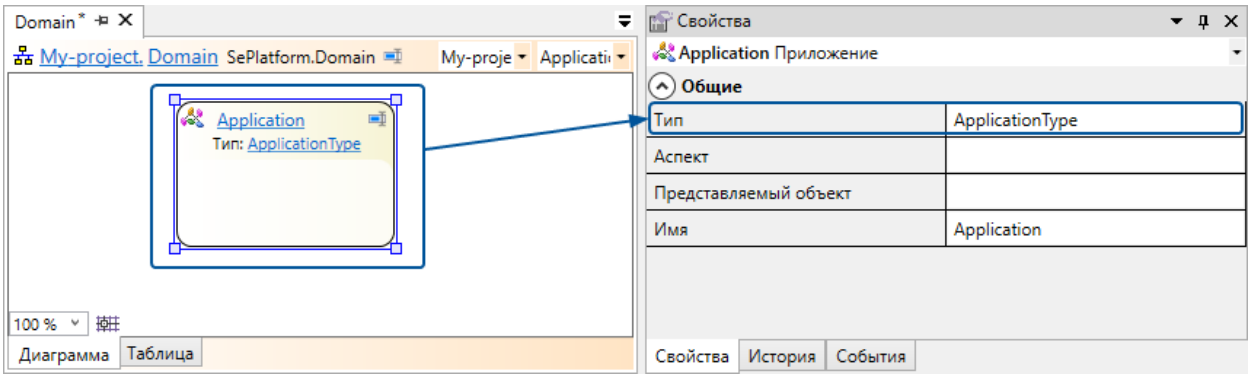



2.3. Если тип приложения создаётся на основе другого типа приложения, укажите его в свойстве **Базовый тип**.




В типе приложения появятся элементы, добавленные в базовый тип.

2.4. У элемента **Приложение** в свойствах укажите созданный тип.




ПРИМЕЧАНИЕ

В приложениях, размещаемых в SePlatform.AccessPoint, вместо добавления объектов рекомендуется использовать привязки (стр. 103).

3. Если в приложении или его объектах есть ссылки, требующие инициализации, инициализируйте их (стр. 67).

После добавления приложений, настройте передачу данных (стр. 48).

## 5.1.4. Настройка передачи данных

ПРИМЕЧАНИЕ

Пример настройки приведён в разделе 3.2. Передача данных (стр. 22)

Данные передаются между компонентами домена, на которых расположены связанные друг с другом объекты. Условием передачи данных является наличие в одном из компонентов ссылки на объект, находящийся в другом компоненте. Передача данных выполняется по одному или нескольким протоколам передачи данных.

| Протокол | Карта адресов | Серверный логический адаптер | Клиентский логический адаптер |
|---|---|---|---|
| TCP[1] | Не требуется | **`TCP Server`** | **`HUB Module`** |
| OPC DA | **`Карта адресов OPC DA`** Требуется только в сторонних компонентах | **`OPC DA Сервер`** | **`OPC DA Клиент`** |
| OPC UA | **`Карта адресов OPC UA`** Требуется только в сторонних компонентах | **`OPC UA Сервер`** | **`OPC UA Клиент`** |

[1]В Систэм Платформ используется внутренний протокол на базе TCP

| Протокол | Карта адресов | Серверный логический адаптер | Клиентский логический адаптер |
|---|---|---|---|
| IEC 6087-5-101<br>(МЭК 870-5-101) | **Карта адресов МЭК 60870-5 (шлюз)** | **Станция МЭК 60870-5-101** | **Опросчик МЭК 60870-5-101** |
| IEC 6087-5-104<br>(МЭК 870-5-104) | **Карта адресов МЭК 60870-5 (шлюз)** | **Станция МЭК 60870-5-104** | **Опросчик МЭК 60870-5-104** |
| IEC 61850<br>(МЭК 61850) | **Карта адресов МЭК 61850** | **Станция МЭК 61850** | **Опросчик МЭК 61850** |
| Modbus TCP | **Карта адресов Modbus** | **Станция Modbus TCP** | **Опросчик Modbus TCP** |
| Modbus RTU | **Карта адресов Modbus** | **Станция Modbus RTU** | **Опросчик Modbus RTU** |
| SNMP | **Карта адресов SNMP** | **Агент SNMP**<br>Может быть добавлен только в сторонний компонент | **Менеджер SNMP** |

Чтобы настроить передачу данных между двумя компонентами по выбранному протоколу:

1. В компонент, который предоставляет или передаёт данные:
   1.1. Добавьте карту адресов (стр. 49).
   1.2. Добавьте серверный логический адаптер (стр. 50).
2. В компонент, который запрашивает или получает данные, добавьте клиентский логический адаптер (стр. 51).
3. Объедините компоненты в сеть (стр. 51).

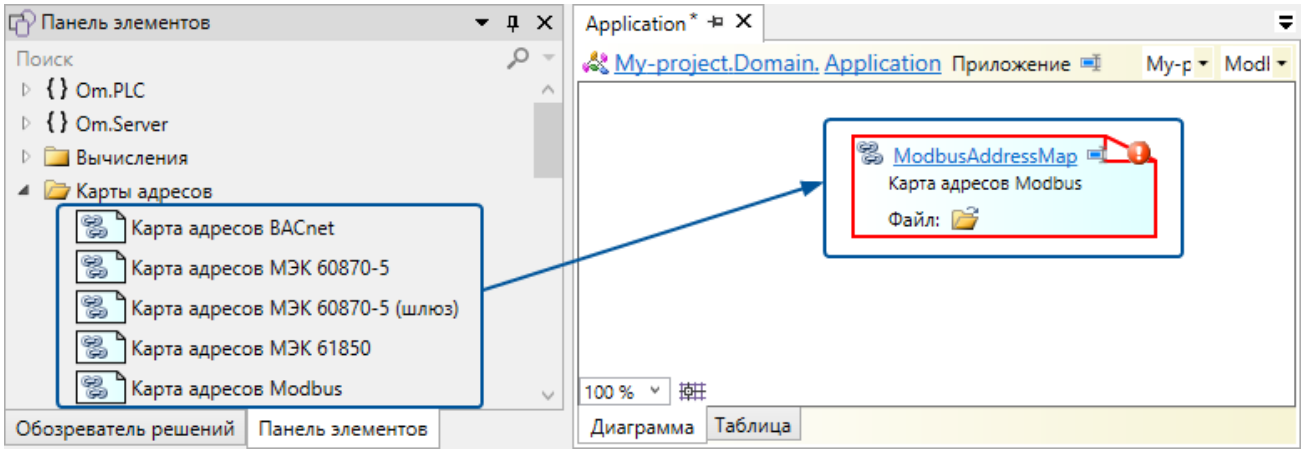
После настройки передачи данных для всех компонентов, проверьте достижимость данных (стр. 55).

## 5.1.4.1. Добавление карты адресов

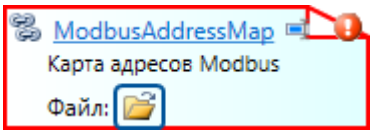
Карта адресов создаётся в приложении. В ней для сигналов приложения указываются их адреса: адреса используются для запроса/предоставления значений сигналов.

Чтобы создать карту адресов:

1. Перейдите в **Приложение**.
2. Добавьте карту адресов.




3. Создайте или выберите файл для хранения адресов:
   3.1. Нажмите на иконку папки на изображении карты адресов.




   Откроется окно выбора файла.
   3.2. Создайте или выберите файл.

В результате:

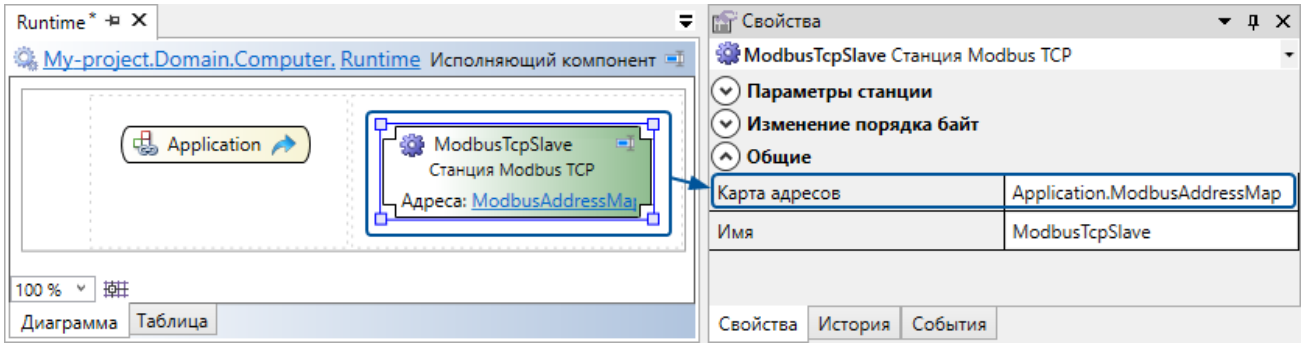
- Файл будет добавлен в список файлов проекта.
- У карты адресов в свойствах будет указан выбранный файл.

4. В карте адресов укажите адреса сигналов.

Параметры адреса зависят от типа карты связывания.

## 5.1.4.2. Добавление серверного логического адаптера

1. Перейдите в компонент.
2. Добавьте в него серверный логический адаптер, указанный в таблице.
3. Если для выбранного протокола используется карта адресов, укажите её в свойствах логического адаптера.



## 5.1.4.3. Добавление клиентского логического адаптера

1. Перейдите в компонент.
2. Добавьте клиентский логический адаптер, указанный в таблице.

## 5.1.4.4. Объединение компонентов в сеть

Чтобы компоненты домена могли обмениваться данными, их нужно объединить в сеть. Есть три типа сетей:

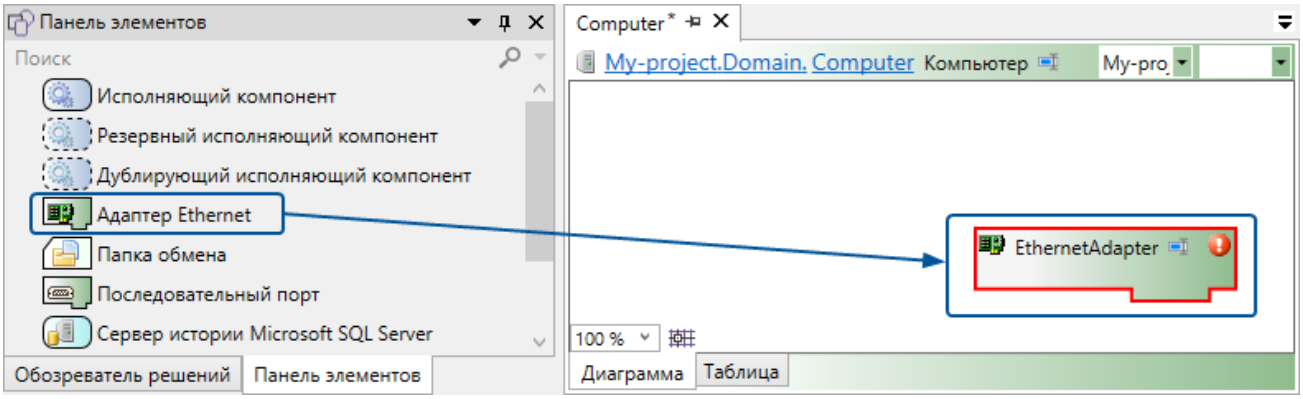
- Ethernet
- Последовательная шина
- Файловый обмен

> ОБРАТИТЕ ВНИМАНИЕ
> Сеть нужно добавлять даже если все компоненты расположены на одном компьютере.

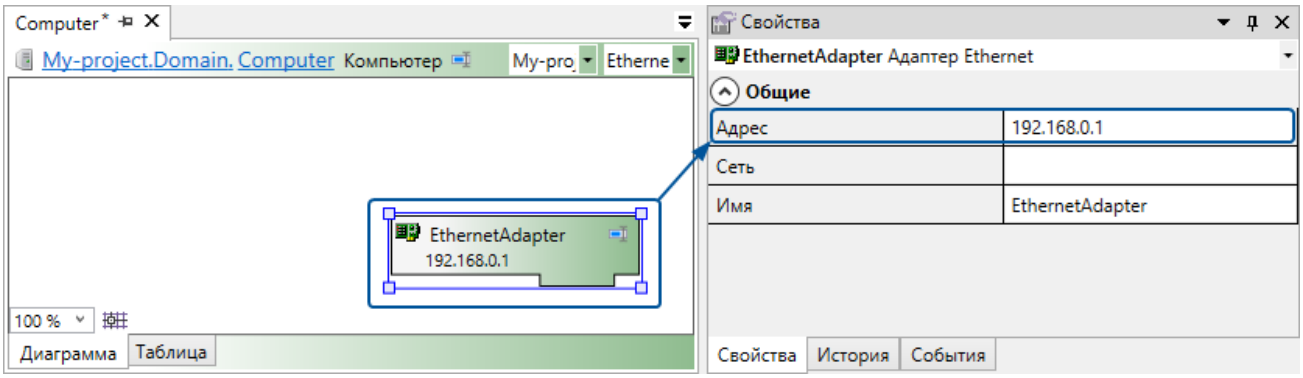
## Сеть Ethernet

Сеть Ethernet - стандарное сетевое соединение между компьютерами, используется для передачи данных по большинству протоколов. В одну сеть включаются все компьютеры и устройства, которые имеют прямое сетевое соединение друг с другом.
Чтобы добавить сеть:

1. В компьютеры, которые нужно объединить в сеть, добавьте **Адаптер Ethernet**.




В свойстве **Адрес** укажите IP-адрес адаптера.

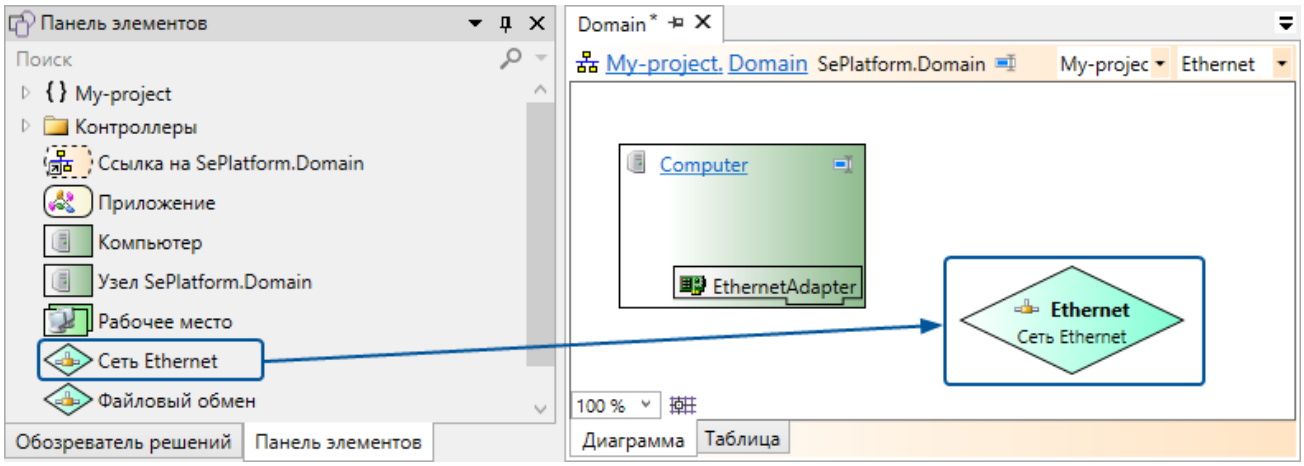



В элементах типа **Рабочее место** в качестве адаптера используется **Клиентский адаптер Ethernet**. У него свойство **Адрес** отсутствует, поскольку элементу может соответствовать любое количество компьютеров.

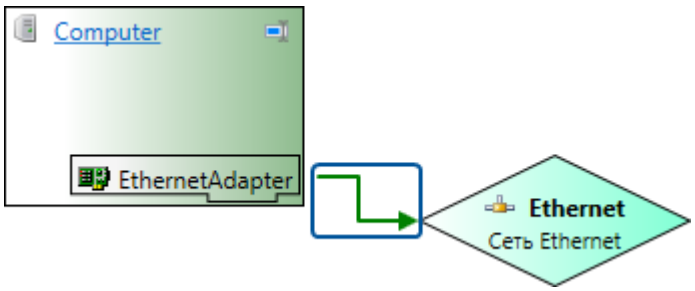
ОБРАТИТЕ ВНИМАНИЕ

Компьютер может иметь несколько адаптеров: разные адаптеры используются для соединения с разными сетями.

2. В элемент **SePlatform.Domain** добавьте элемент **Сеть Ethernet**.




3. Соедините компоненты с добавленной сетью.



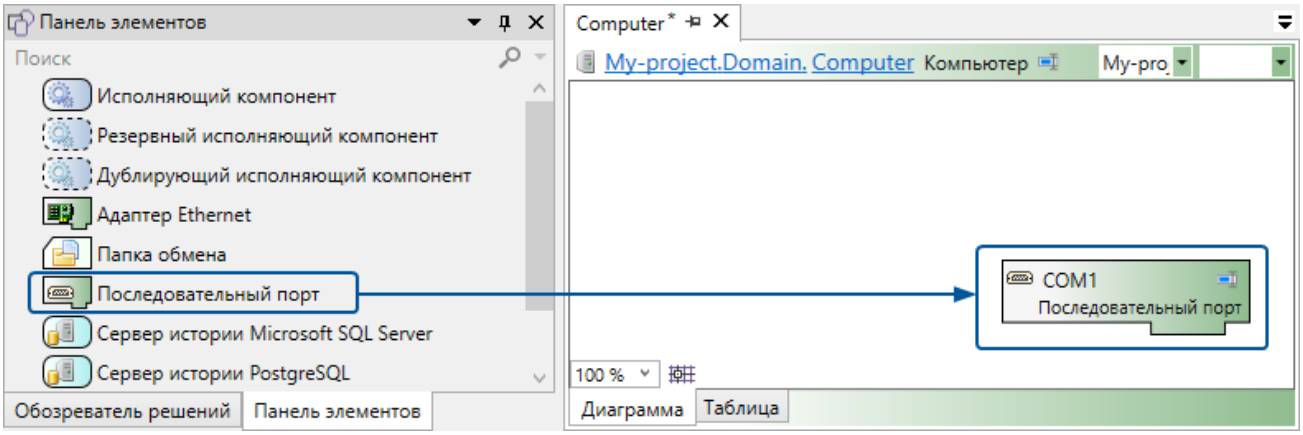

ПРИМЕЧАНИЕ

После соединения у элементов **Адаптер Ethernet** и **Клиентский адаптер Ethernet** в свойстве **Сеть** будет указан связанный элемент.

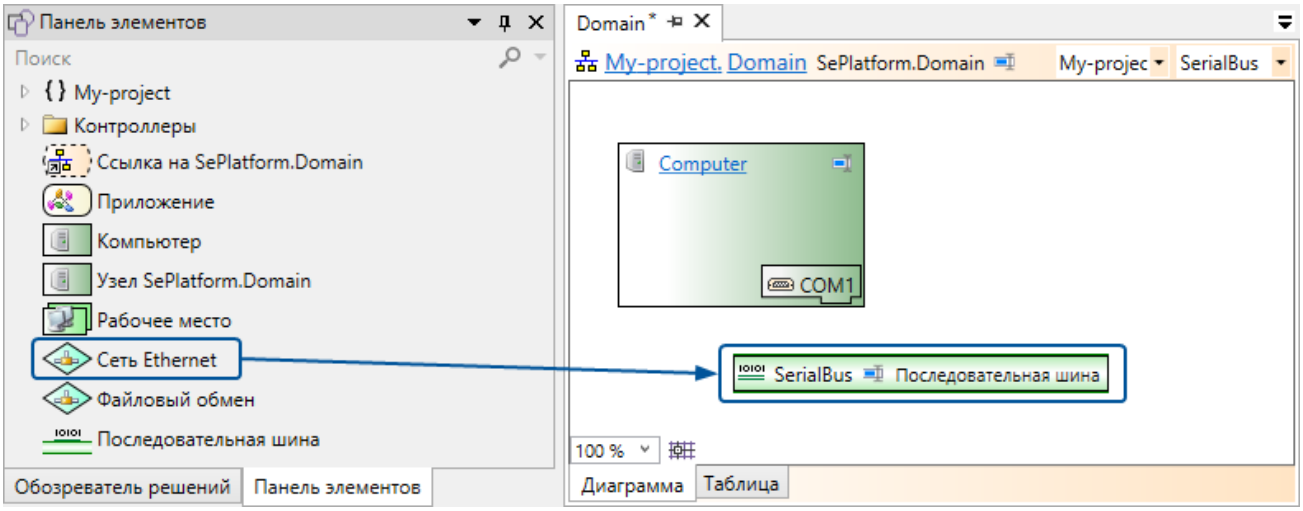
## Последовательная шина

Последовательная шина - физический интерфейс, по которому данные передаются последовательно бит за битом, используется для передачи данных по протоколам Modbus RTU, МЭК 870-5-101. Одна последовательная шина соединяет один опросчик и одну или несколько станций, которые он опрашивает. Чтобы добавить последовательную шину:

1. В компьютеры, которые должны находится на шине, добавьте **Последовательный порт**.

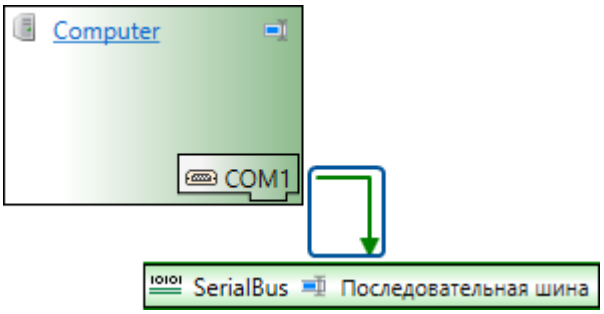



2. В элемент **SePlatform.Domain** добавьте элемент **Последовательная шина**.




В свойствах укажите параметры шины: скорость, чётность, количество стоповых битов и способ управления потоком.

3. Соедините компьютеры с добавленной шиной.




ПРИМЕЧАНИЕ

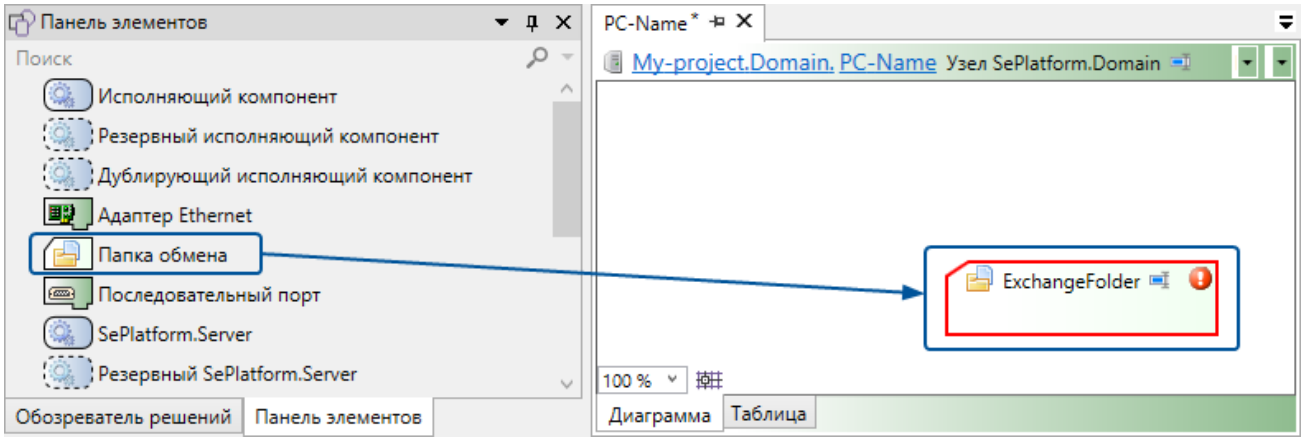
После соединения у элементов **Последовательный порт** в свойстве **Шина** будет указан связанный элемент.

## Файловый обмен

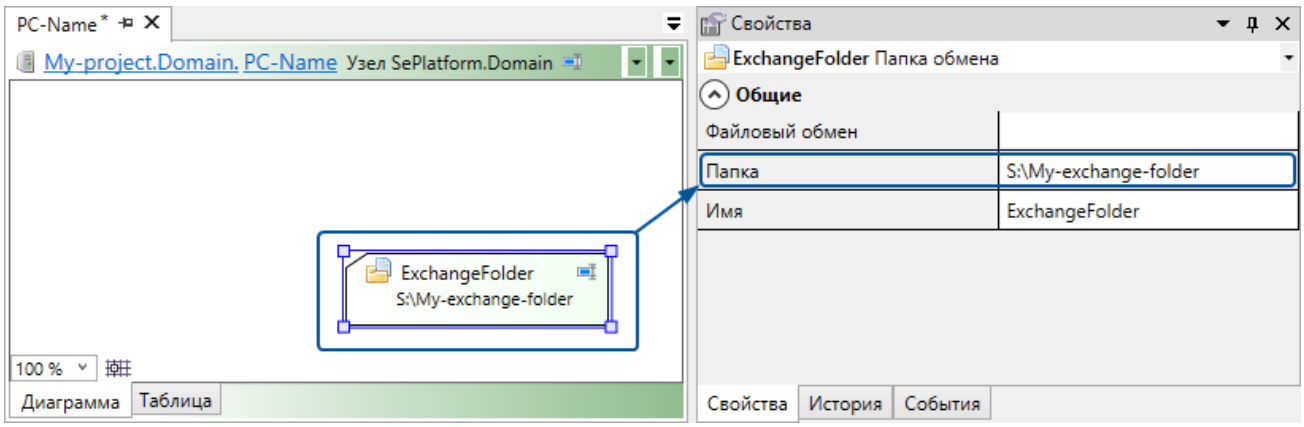
Файловый обмен соединяет пару компонентов (SePlatform.Data Server - SePlatform.AccessPoint или SePlatform.Data Server - SePlatform.Data Server), в которых модули TCP Server и HUB используют файловый интерфейс для обмена данными: модуль TCP Server записывает данные в файлы в сетевой папке, а модуль HUB считывает эти файлы.

Чтобы настроить файловый обмен:

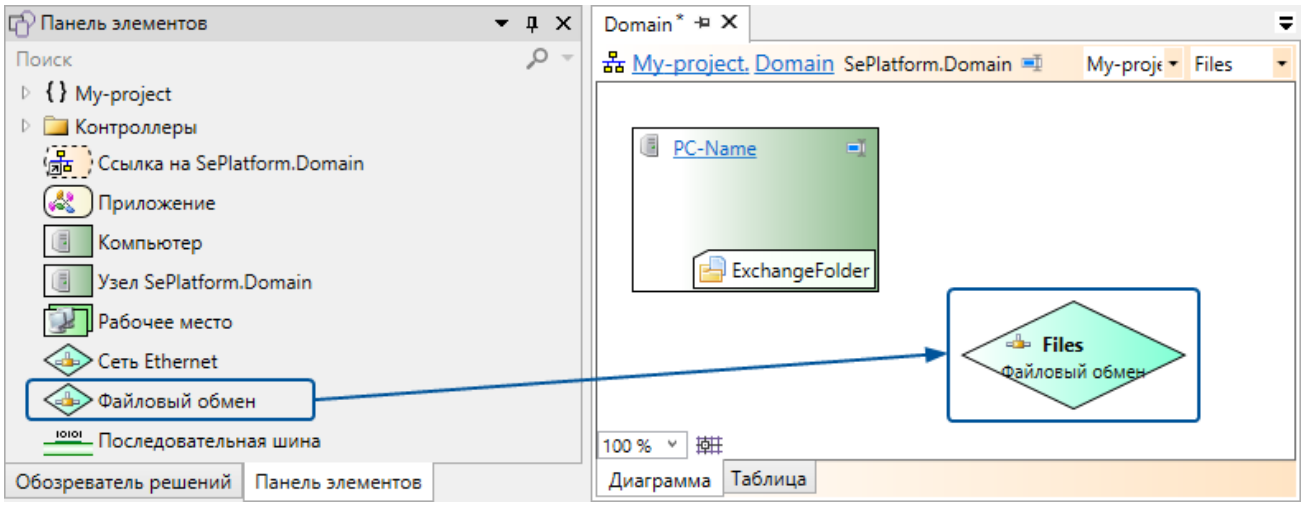
1. В компьютеры, между которыми передаются данные, добавьте элемент **Папка обмена**.



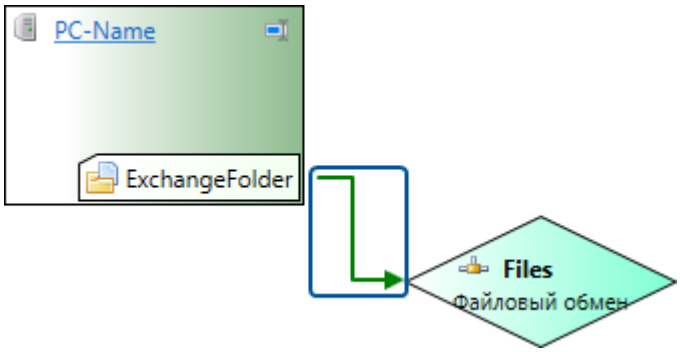

В свойстве **Папка** укажите путь к сетевой папке.




2. В элемент **SePlatform.Domain** добавьте элемент **Файловый обмен**.



3. Соедините компьютеры с добавленным элементом.




ПРИМЕЧАНИЕ

После соединения у элементов **Папка обмена** в свойстве **Файловый обмен** будет указан связанный элемент.

## 5.1.4.5. Проверка достижимости данных

Чтобы проверить, что данные будут успешно передаваться между компонентами, постройте решение.


При построении SePlatform.Development Studio проверит достижимость данных.

ПРИМЕЧАНИЕ

Обозначения в тексте сообщений:

- {runtime} - исполняющий компонент;
- {adapter} - логический адаптер;
- {member} - член объекта с данными.

1. Вначале проверяется наличие связи между исполняющими компонентами.

Для передачи данных требуется наличие сети и подключенных к ней физических адаптеров, а также требуется наличие совместимых логических адаптеров. Если связи между исполняющими компонентами отсутствуют, то будут сообщения об ошибке:

- У {runtime} нет адаптеров, которые могут быть использованы для соединения с удаленными исполняющими компонентами

  То есть у исполняющего компонента вообще нет каких-либо клиентских логических адаптеров

- У {runtime} нет соединений с исполняющими компонентами через имеющиеся клиентские адаптеры

  То есть у исполняющего компонента есть клиентские логические адаптеры, но ни один из них не имеет соединения с серверным логическим адаптеров на каком-либо исполняющем компоненте.

2. Если имеются связи с другими исполняющими компонентами, то проверяется наличие нужных данных, размещенных в этих исполняющих компонентах. В случае отсутствия данных будет сообщение об ошибке:

- У {runtime} есть соединения с исполняющими компонентами, но они не содержат необходимые данные

  То есть во всех доступных исполняющих компонентах нет приложения или исполняемого модуля с объектами, на которые имеются ссылки в данном исполняющем компоненте.

3. Одного размещения объектов на удаленных исполняющих компонентах недостаточно. Требуется серверный логический адаптер, который может предоставить эти данные по определенному протоколу. Если таких адаптеров нет, то будет сообщение об ошибке:

› У {runtime} есть соединения с исполняющими компонентами, но удаленные серверные адаптеры не могут предоставить нужные данные

Например, когда исполняемый модуль приложения привязан к другому серверному адаптеру.

4. Дальнейшие проверки выполняются для каждого члена объекта с данными. Дело в том, что некоторые члены объекта не могут быть переданы по имеющемуся соединению. Это происходит, например, при передаче с разделением потоков данных с помощью категорий. В зависимости от количества имеющихся соединений с исполняющими компонентами будут следующие ошибки:

› {member} не будет доступен в {runtime}, так как не привязан к какому либо серверному адаптеру {adapter}

То есть данные члена объекта не могут быть предоставлены каким-либо логическим адаптером, через которые есть соединения с этим удаленным исполняющим компонентом;

› {member} не привязан ни к одному из соединений {runtime} с другими исполняющими компонентами

То есть среди всех имеющихся соединений не нашлось серверного адаптера, способного предоставить данные этого члена объекта.

5. И наконец, может быть корректное соединение, серверный адаптер может предоставить данные по нужному члену объекта, но в карте адресов этого адаптера нет описания связи (адреса сигнала). Данные в таком случае передаваться не будут. Но и ошибкой это считать нельзя, так как в процессе разработки проекта автоматизации может потребоваться построение и отладка сервера с частично известными адресами. Поэтому будут предупреждения:

› {member} не будет доступен в {runtime}, так как нет какого-либо описания связи с другими исполняющими компонентами

› {member} не будет доступен в {runtime}, так как отсутствует описание связи у {adapter}

# 5.2. Добавление объектов

Объект - это набор данных и правил работы с ними. Объекты размещаются в компонентах домена (экземплярах SePlatform.Data Server и службах и устройствах, с которыми они взаимодействуют) и описывают данные, с которыми эти компоненты работают.

С помощью объектов описываются реальные или абстрактные сущности: заводы, трубопроводы, датчики и т.д.. Несколько объектов могут описывать одну и ту же сущность. Например, объекты **«Датчик в ПЛК»** и **«Датчик в SePlatform.Data Server»** описывают один датчик, показания которого передаются из ПЛК в SePlatform.Data Server. Для указания, что несколько объектов описывают одну и ту же сущность, используются аспекты.

Аспект - это некоторый угол зрения. Каждая сущность описывается в одном или нескольких аспектах (под несколькими углами зрения): в каждом аспекте сущность представляет собой объект. Объекты, которые описывают одну сущность в разных аспектах, называются представлениями.

Если сущность описывается в нескольких аспектах, сначала создаётся её представление (объект) в одном аспекте, а для других представлений (объектов) этой сущности указываются представляемый объект и аспект. Например, у объекта **«Датчик в SePlatform.Data Server»** будет указано, что он представляет объект **«Датчик в ПЛК»** в аспекте **«SePlatform.Data Server»**: это означает, что оба объекта описывают один и тот же датчик, но в разных аспектах.

ОБРАТИТЕ ВНИМАНИЕ

В приведённом выше примере каждый аспект указывает на определённый тип компонента: ПЛК или SePlatform.Data Server. Но эти термины неидентичны: в одном компоненте можно разместить объекты в разных аспектах.

Например, пусть **«Датчик в ПЛК»** передаёт не только измеренные значения, но и статистику о своей работе. Чтобы разделить эти два типа информации, можно создать два объекта: **«Датчик»** в аспекте **«Сервер ввода/вывода»** и **«Датчик»** в аспекте **«Статистика»**. Эти объекты можно разместить:

- в двух разных экземплярах SePlatform.Data Server: тогда один SePlatform.Data Server будет принимать и обрабатывать значения датчика, а другой – статистику его работы.
- в одном экземпляре SePlatform.Data Server: тогда один SePlatform.Data Server будет принимать и обрабатывать как значения датчика, так и его статистику.

Для передачи данных между представлениями используются аспектные ссылки. Аспектная ссылка – это ссылка, которая добавляется в объект и указывает на другое представление этого объекта.

ПРИМЕЧАНИЕ

Для просмотра представлений объекта и связей между ними используется схема представлений (стр. 179).

Объекты можно описывать с помощью типов: в этом случае содержимое объекта описывается в типе, а объекту указывается его тип. Типы облегчают создание и изменение однотипных объектов.

Порядок добавления объектов:

1. (Рекомендуется) Предварительно создайте аспекты (стр. 57).
2. Опишите объекты или их типы (стр. 58).
3. Если описаны типы, разместите в приложениях объекты описанных типов (стр. 84).

ПРИМЕЧАНИЕ

Аспекты используются для быстрого размещения объектов, для которых описаны типы. Если типы не используются, создавать и использовать аспекты не обязательно, поскольку в этом случае все объекты создаются и описываются вручную.

## 5.2.1. Создание аспектов

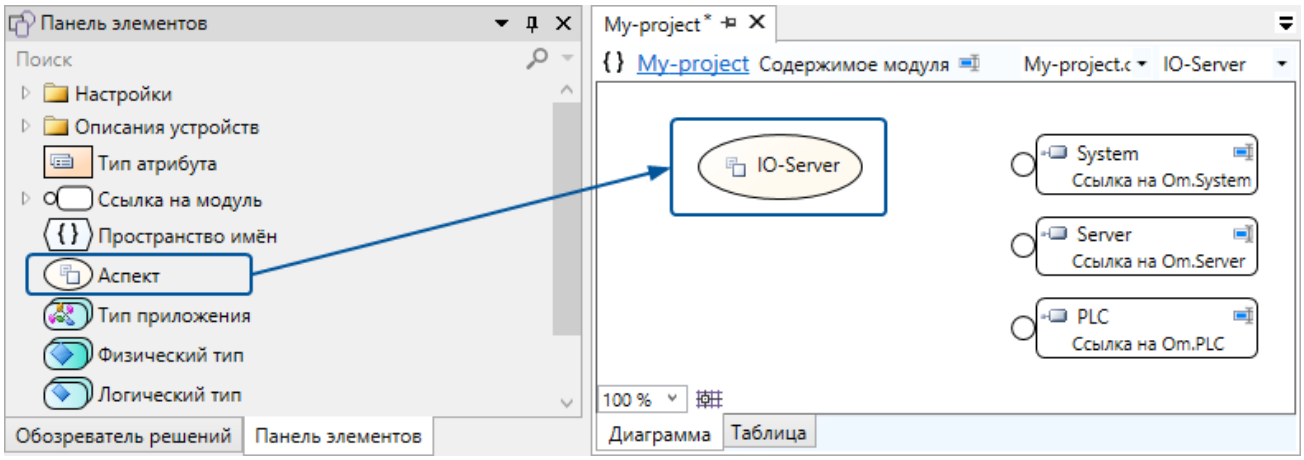
Чтобы создать аспект:

1. Перейдите в **Содержимое модуля** (корень проекта), **Пространство имён** или в **Тип приложения**.

ПРИМЕЧАНИЕ

Аспект можно добавлять также внутрь типов, однако мы рекомендуем этого не делать, чтобы не создавать путаницы. Будет казаться, будто аспект принадлежит типу, в котором находится, хотя на самом деле этот аспект может использоваться для любых типов и объектов.

2. Добавьте **Аспект** и укажите ему имя.




После создания аспекта, можно описывать объекты и типы в этом аспекте.

## 5.2.2. Описание объектов

Описание объекта - это описание содержимого объекта. Содержимое объекта описывается в типе (рекомендуется) или непосредственно в объекте. Создание объектов, для которых описаны типы, приведено в разделе 5.2.3. Размещение объектов (стр. 84).

Чтобы описать объект:

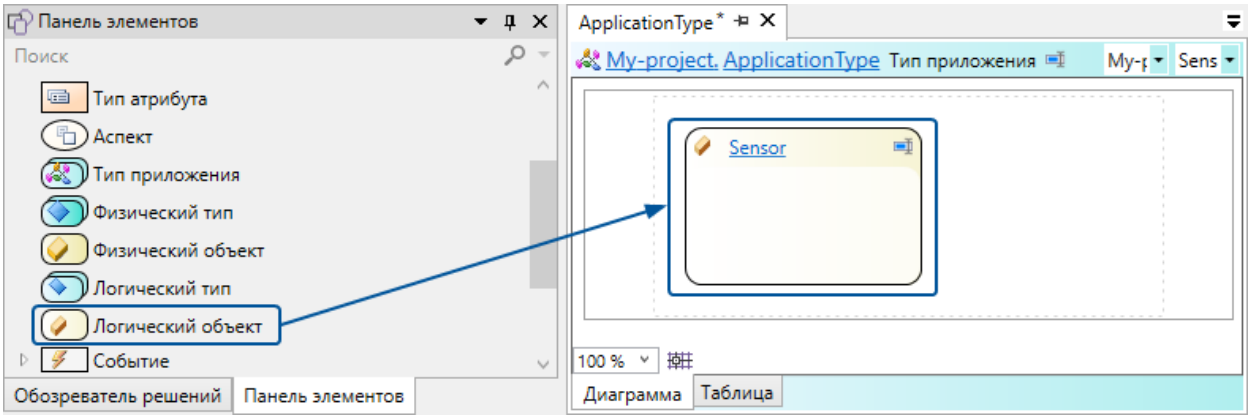
1. Создайте объект или тип.

Создание объекта:

1.1. Перейдите в элемент в котором хотите создать объект:

- **Приложение** - объект будет размещён в этом приложении.
- **Тип приложения** - объект будет размещён в каждом приложении этого типа.
- **Логический объект** - объект будет вложен в этот объект.
- **Логический тип** - объект будет вложен в каждый объект этого типа.

1.2. Добавьте **Логический объект**.




Создание типа:

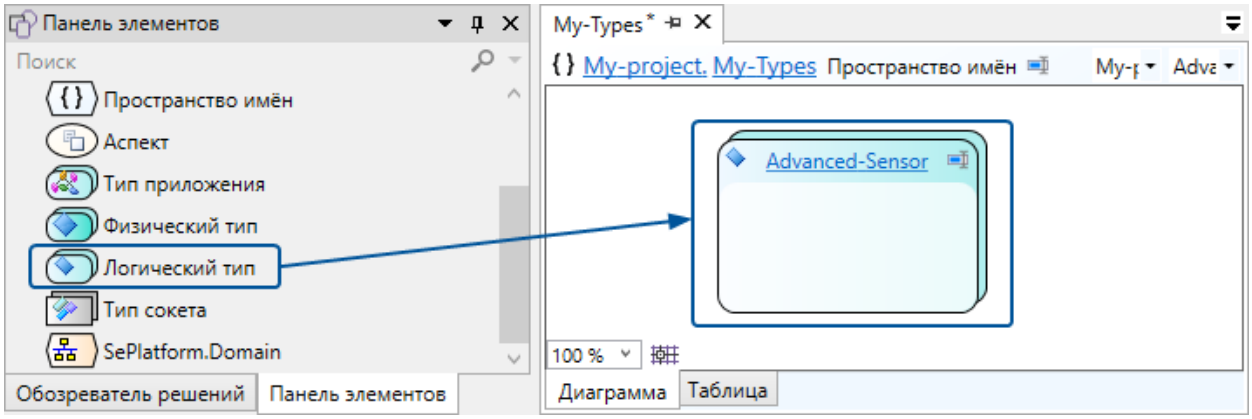
1.1. Перейдите в **Пространство имён**, в котором хотите создать тип.

ПРИМЕЧАНИЕ

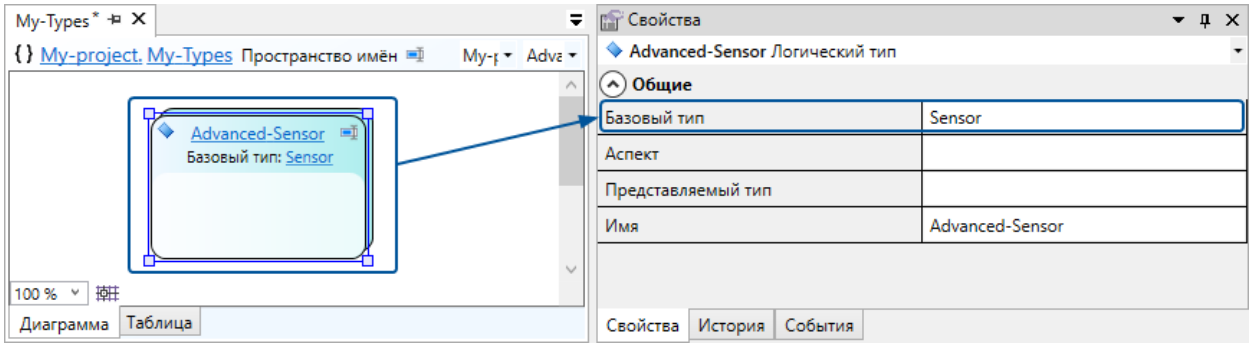
Тип можно добавлять также внутрь других элементов: типов приложений и типов, однако мы рекомендуем этого не делать, чтобы не создавать путаницы. Визуально будет казаться, что в элементе размещён элемент с данными, хотя на самом деле тип - это только описание, поэтому в элементе этих данных не будет.

Также рекомендуем не добавлять типы в корень проекта, чтобы не загромождать основное рабочее пространство проекта.

1.2. Добавьте **Логический тип**.




1.3. Если тип создаётся на основе другого типа, укажите его в свойстве **Базовый тип**.

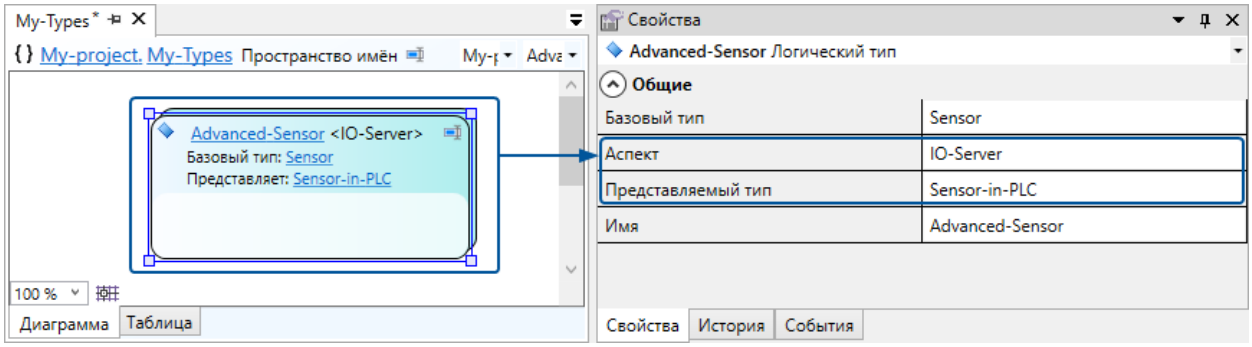



В типе появятся элементы, добавленные в базовый тип.

1.4. Если тип представляет другой тип:

1.4.1. В свойстве **Представляемый тип** укажите тип, для которого данный тип является представлением.

1.4.2. В свойстве **Аспект** укажите аспект текущего типа.



2. Опишите содержимое объекта:
    2.1. Добавьте сигналы (стр. 60).
    2.2. Добавьте вложенные объекты и ссылки (стр. 63).
    2.3. Добавьте связи (стр. 69).
    2.4. Добавьте вычисления (стр. 74).
    2.5. Настройте генерацию событий (стр. 80)

## 5.2.2.1. Добавление сигналов

В SePlatform.Development Studio есть несколько элементов для описания сигналов:

- Параметры
- События

Параметры - описывают характеристики объекта в любой момент времени. Могут быть постоянны на всём протяжении существования объекта. Имеют направление:

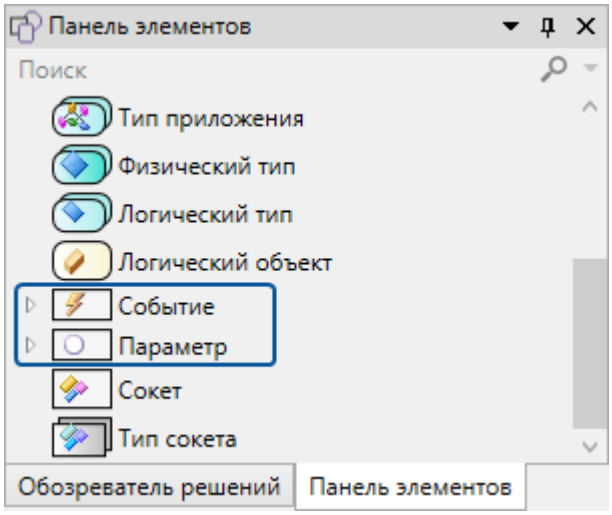
- входные - параметры, приходящие извне.
- выходные - параметры, принадлежащие объекту.

События - изменяют состояние объекта и существуют только в момент возникновения.
Например, передаваемые команды: **«Открыть»**, **«Включить»**. Имеют направление:

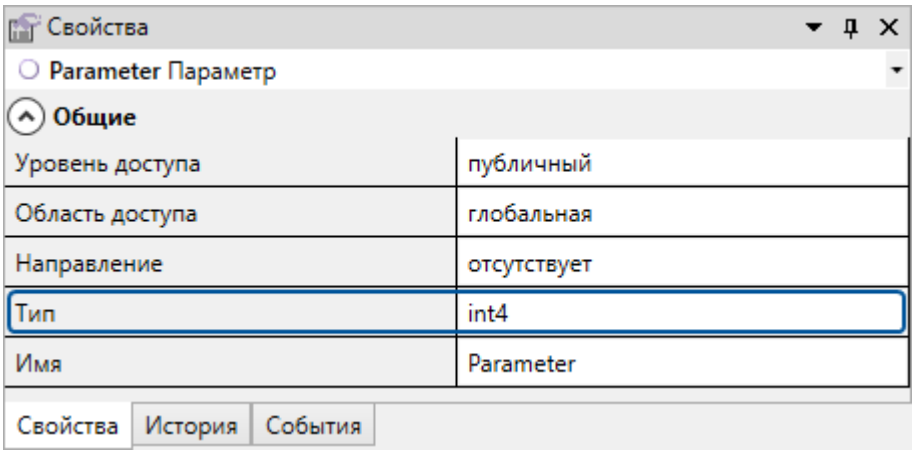
- входные - события, которые вызывают изменение значений параметров объекта.
- выходные - события, которые возникают при изменениях параметров объекта.

Чтобы добавить сигнал:

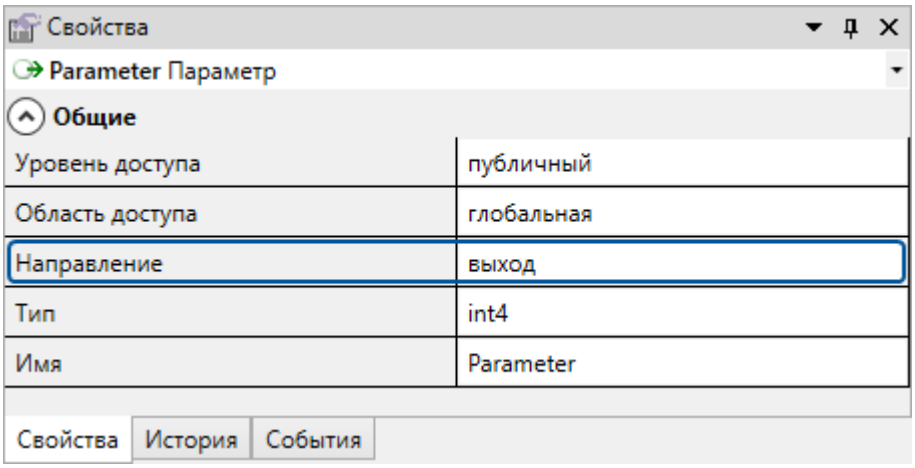
1. Добавьте элемент нужного типа:
    - **Параметр**
    - **Событие**



2. В свойстве **Тип** укажите тип значений.




3. В свойстве **Направление** укажите, является ли сигнал входным или выходным для объекта.

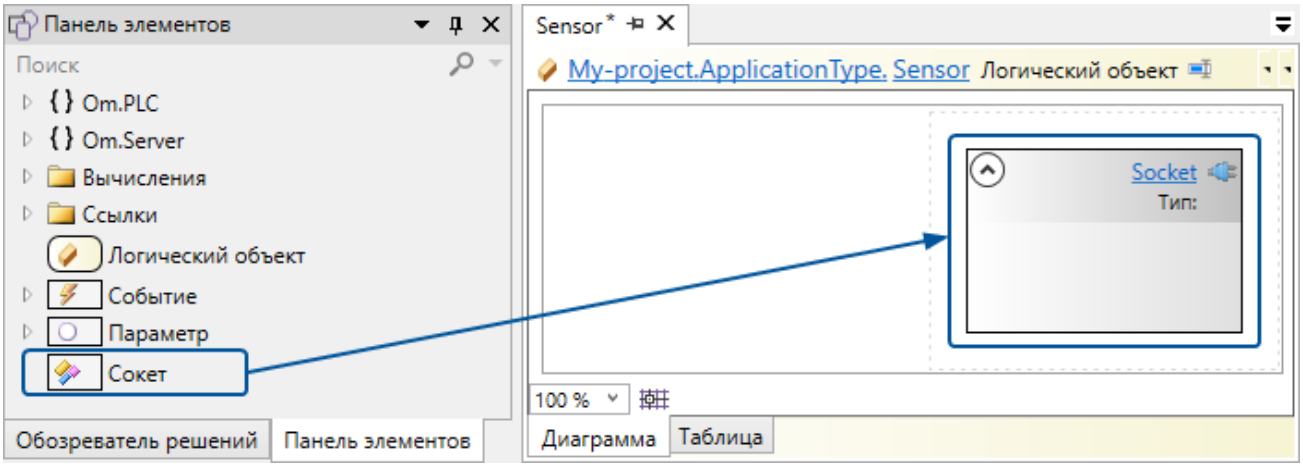



## Сокеты

Для описания сигналов, чьи значения передаются из одного объекта в другой, рекомендуется использовать сокеты. Сокет - это группа сигналов. При использовании сокетов связь добавляется не между отдельными сигналами, а между сокетами: входящие в них сигналы связываются автоматически.

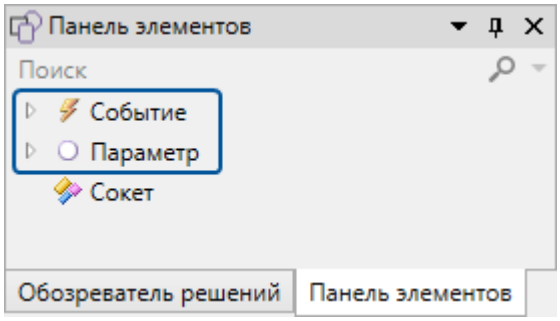
Чтобы создать сокет:

1. Добавьте **Сокет**.




В свойстве **Направление** укажите, является ли сокет входным или выходным для объекта.

2. Перейдите в **Сокет**.
3. Добавьте сигналы нужного типа:
   - **Параметр**
   - **Событие**




Каждому сигналу в свойствах укажите его тип значений.

ПРИМЕЧАНИЕ

Помимо сигналов, в сокет можно добавлять вложенные сокеты.

ОБРАТИТЕ ВНИМАНИЕ

При построении конфигураций, сокет не преобразуется в папку дерева сигналов. Вместо этого сигналы сокета попадут в папку объекта, в котором находится сокет. Это значит, что имена сигналов в объекте и в сокете не должны совпадать.

Чтобы использовать сокет в нескольких объектах/типах, можно создать тип сокета. Использование типа сокета позволяет описывать сигналы в одном месте: при добавлении сигнала в тип сокета, этот сигнал появится у каждого сокета этого типа.

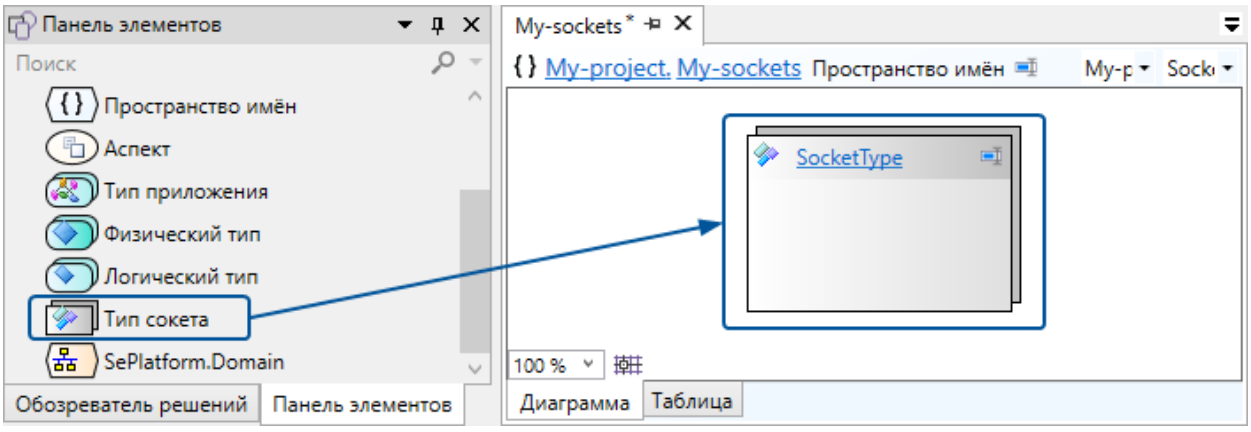
Чтобы использовать тип сокета:

1. Создайте тип сокета:
   1.1. Перейдите в **Пространство имён**, в котором хотите описать тип сокета.

ПРИМЕЧАНИЕ

Типы сокетов можно описывать также внутри других элементов, однако мы рекомендуем этого не делать, чтобы не создавать путанницы. Будет казаться, что в элементе находятся сигналы, описанные в типе сокета, хотя на самом деле тип сокета - это только описание, поэтому в элементе этих данных не будет.

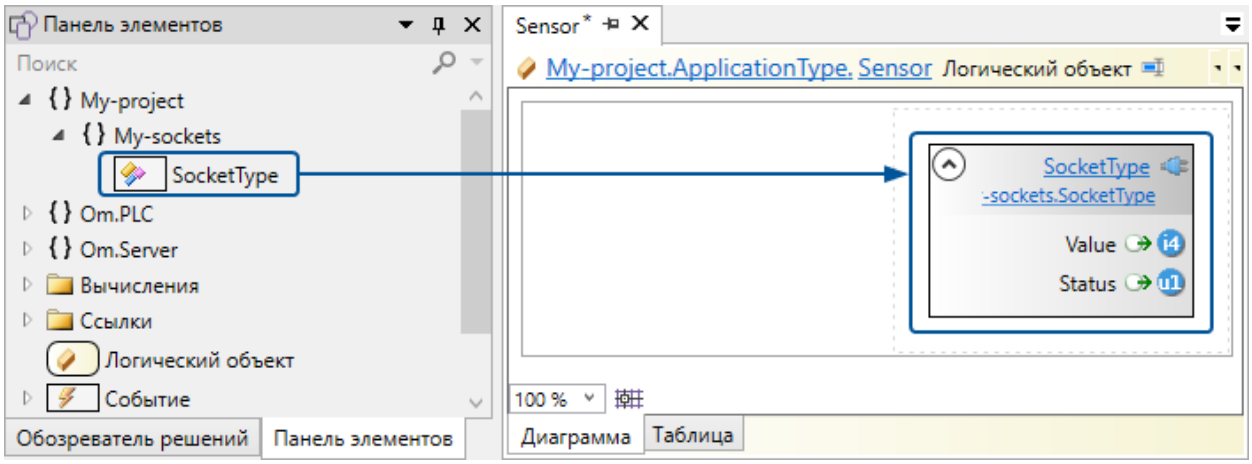
   1.2. Добавьте **Тип сокета**.



1.3. В **Тип сокета** добавьте сигналы: выполняется так же, как при добавлении сигналов в сокет.

2. В объект/тип добавьте сокет созданного типа:

2.1. В панели элементов раскройте узел с именем проекта и перетащите из него сокет созданного типа в рабочую область. Сокет находится по тому же пути, что и его тип в проекте.




2.2. В свойстве **Направление** укажите направление сокета:

- **«Вход»**
- **«Выход»**

## 5.2.2.2. Добавление вложенных объектов и ссылок

### Вложенные объекты

Вложенный объект - это объект, который находится внутри другого объекта. Если объект вложен в тип, внутри каждого объекта этого типа будет находиться свой экземпляр вложенного объекта. На изображении вложенного объекта отображаются его сигналы: их можно использовать при добавлении связей и в вычислениях.
Способы добавления вложенных объектов:

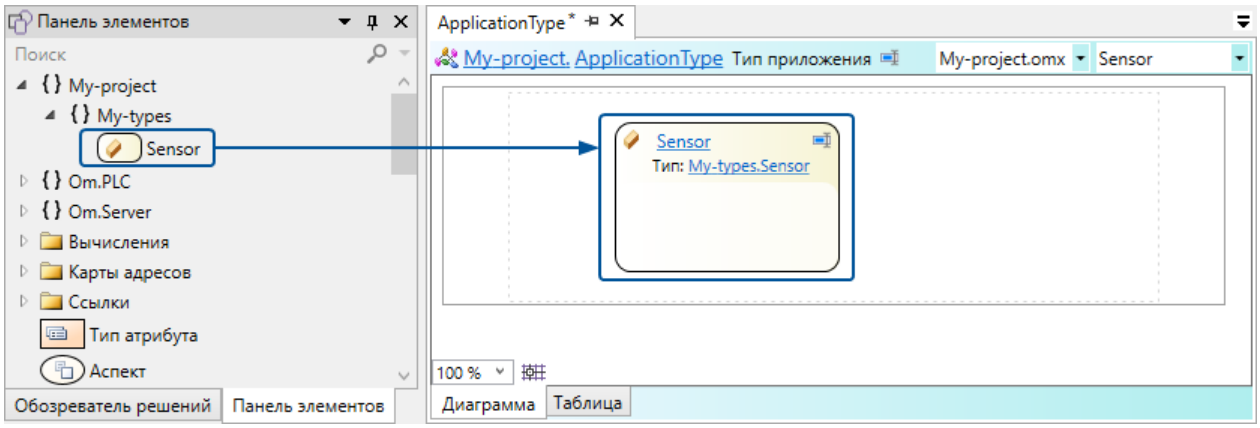
- создать вручную.
- отобразить объекты с помощью мастера. Подробнее см. раздел 5.2.3. Размещение объектов (стр. 84).

Чтобы добавить вложенный объект вручную:

1. Добавьте **Логический объект**.
2. В свойстве **Тип** укажите тип объекта (если есть).

ПРИМЕЧАНИЕ

Чтобы быстро создать объект, для которого описан тип, в панели элементов раскройте узел с именем проекта и перетащите объект описанного типа в рабочую область. Объект находится по тому же пути, что и его тип в проекте.




## Ссылки

Ссылка указывает на объект, который размещён вне объекта или типа, в котором находится ссылка. При размещении объектов в домене, если объект и ссылка на него окажутся в разных компонентах, то данные будут передаваться между этими компонентами по протоколам и интерфейсам, описанным в домене.

ПРИМЕР

При описании домена:

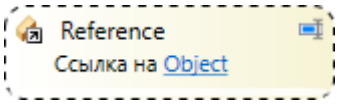
- в ПЛК создан объект **«Датчик»**.
- в SePlatform.Data Server создан объект **«Данные датчика»** и в него добавлена ссылка на объект **«Датчик»**.

В результате:

- значения выходных сигналов объекта **«Датчик»** будут передаваться из ПЛК в SePlatform.Data Server .
- значения входных сигналов объекта **«Данные датчика»** (например, команды) будут передаваться из SePlatform.Data Server в ПЛК.

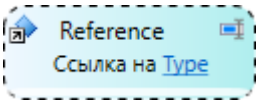
Виды ссылок:

- объектная ссылка




Объектная ссылка указывает на объект, который находится в том же адресном пространстве, что и ссылка. Элементы в которых создаётся собственное адресное пространство: **SePlatform.Domain**, **Тип приложения** и **Логический тип**. Объекты, которые в них вложены (любого уровня вложенности), находятся в одном адресном пространстве.

- ссылка




Ссылка используется при описании типа и указывает на некоторый объект определённого типа: в каждом объекте описанного типа в ссылку будет подставлен свой объект указанного типа. Подстановка объекта в ссылку называется инициализацией ссылки.

ПРИМЕР

Описаны два типа: **«Датчик в ПЛК»** и **«Датчик в сервере»**. В типе **«Датчик в сервере»** есть ссылка на **«Датчик в ПЛК»**.

При описании домена:

- в ПЛК созданы два объекта типа **«Датчик в ПЛК»**: **«Датчик 1»** и **«Датчик 2»**.
- в SePlatform.Data Server созданы два объекта типа **«Датчик в сервере»**: **«Данные датчика 1»** и **«Данные датчика 2»**.

  В объекте **«Данные датчика 1»** ссылка инициализирована объектом **«Датчик 1»**, в объекте **«Данные датчика 2»** - объектом **«Датчик 2»**.

В результате данные объекта **«Датчик 1»** будут передаваться в объект **«Данные датчика 1»**, а данные объекта **«Датчик 2»** - в объект **«Данные датчика 2»**.

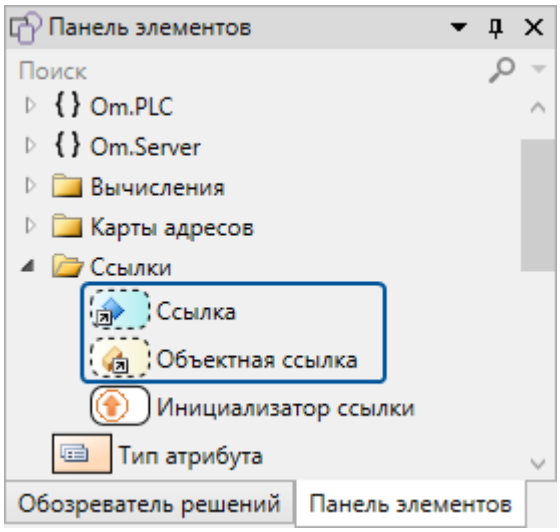
Ссылки обоих видов могут быть аспектными или нет. Аспектная ссылка означает что объект, в котором находится ссылка и объект, на который она указывает, являются представлениями друг друга - описывают одну и ту же реальную или абстрактную сущность. Аспектные ссылки используются для передачи данных между различными представлениями одного объекта. В примере выше ссылка в типе **«Датчик в сервере»** является аспектной: она используется для передачи данных датчика из одного представления в другое.

ПРИМЕЧАНИЕ

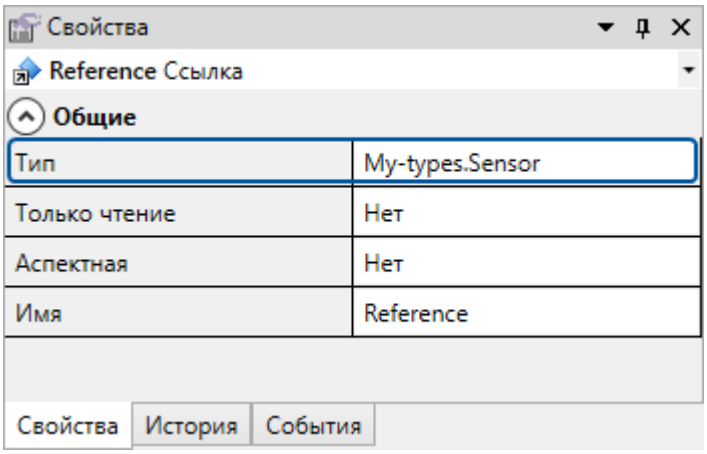
Аспектные ссылки отображаются на схеме представлений типа/объекта (стр. 179).

Способы добавления ссылки:

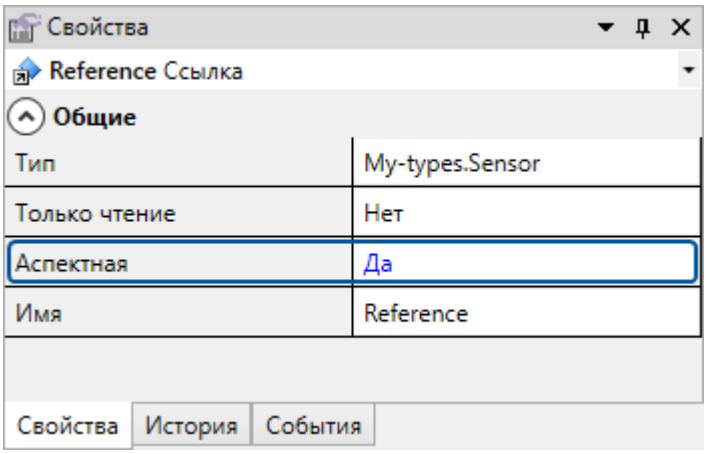
- Вручную:
  1. Добавьте элемент нужного типа:
     - **Объектная ссылка**
     - **Ссылка**



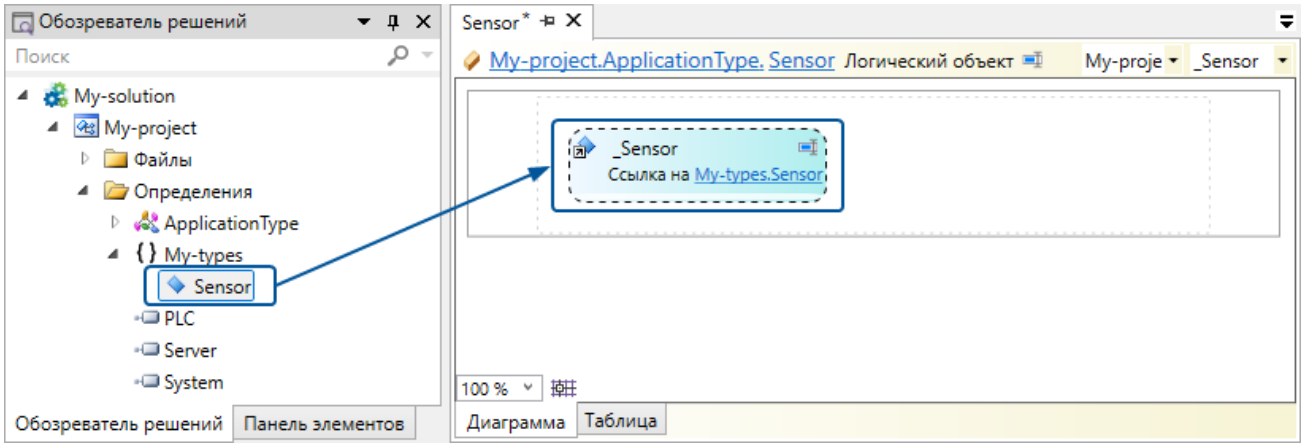
2. В свойствах укажите объект/тип, на который указывает ссылка.



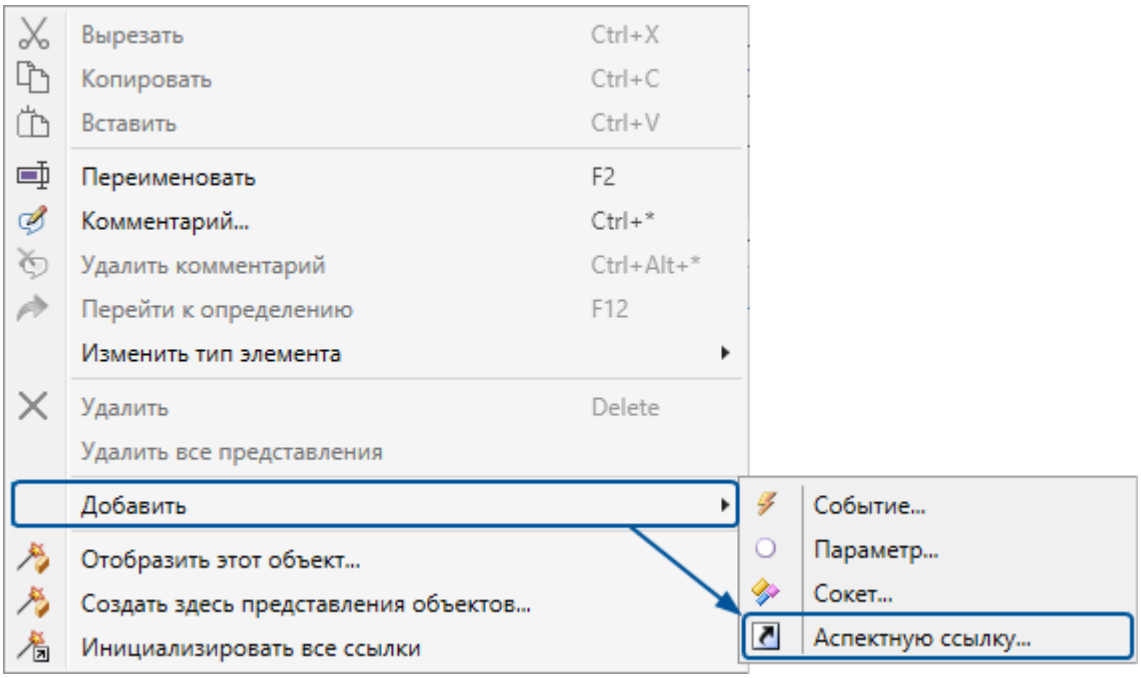

3. В свойстве **Аспектная** укажите, является ли ссылка аспектной.



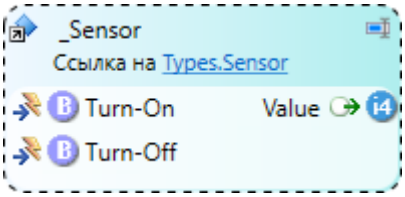

› Быстрое добавление ссылки на объект/тип: перетащите объект/тип из обозревателя решений в рабочую область.



› аспектная ссылка: в контекстном меню объекта/типа выберите **Добавить** → **Аспектную ссылку...** - откроется мастер создания аспектной ссылки.



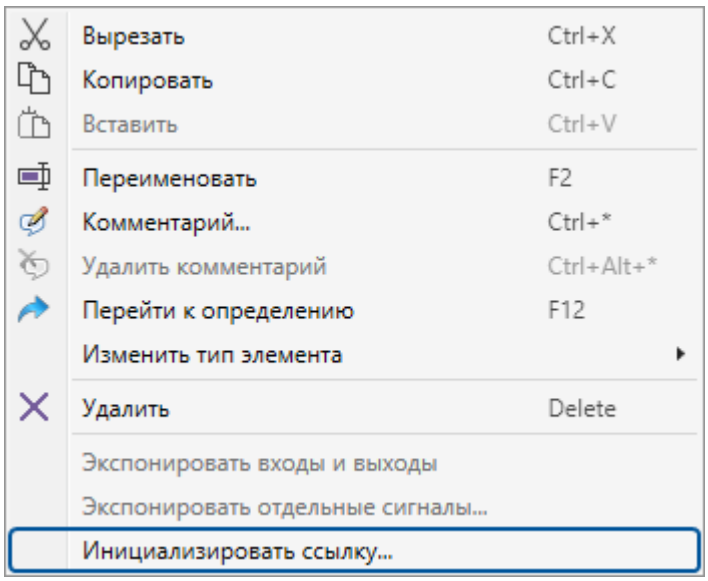

Ссылка принимает вид объекта/типа, на который она ссылается: на изображении ссылки появятся сигналы, которые есть у объекта/типа. Их можно связывать с другими сигналами в объекте/типе.




## Инициализация ссылок

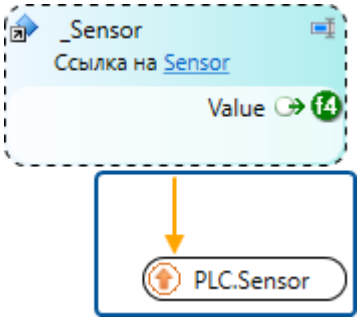
Инициализация ссылки - это указание объекта, который будет подставлен на место ссылки.
Инициализировать ссылку можно объектом того типа, на который указывает ссылка, или его подтипом. При инициализации ссылки в объект добавляется элемент **Инициализатор ссылки**, который связывает ссылку с конкретным объектом.
Чтобы инициализировать ссылку, выполните следующие действия:

1. В контекстном меню ссылки выберите **Инициализировать ссылку...**.

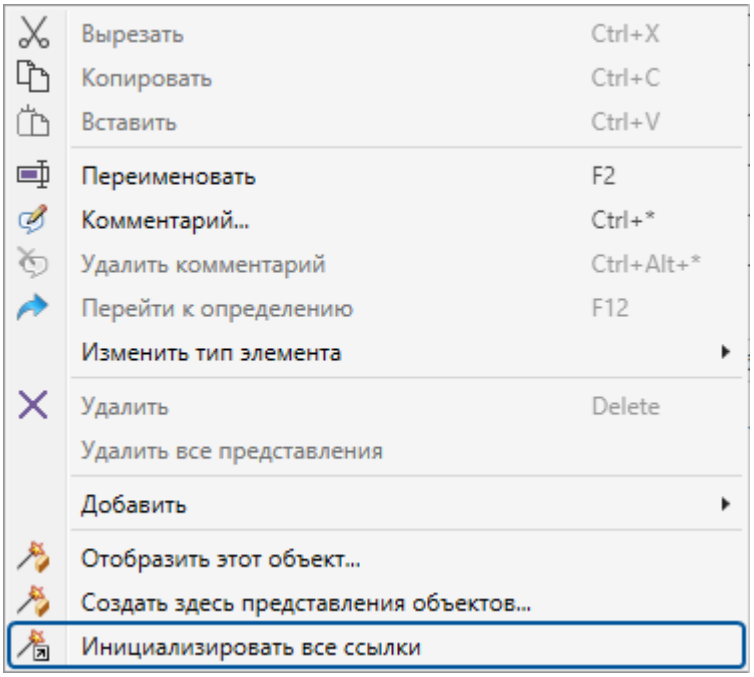



2. В появившемся окне выберите объект, которым необходимо инициализировать ссылку.

В результате ссылка будет инициализирована выбранным объектом.




Чтобы инициализировать ссылки во всех вложенных объектах, в контекстном меню объекта/приложения выберите **Инициализировать все ссылки**.



В результате:

- аспектные ссылки будут инициализированы соответствующими им объектами (если их возможно определить).
- для всех остальных ссылок будут созданы пустые инициализаторы ссылок. В **журнале** появятся сообщения о наличии пустых инициализаторов.

У каждого пустого инициализатора нужно в свойстве **Цель** выбрать объект, с которым он связывает соответствующую ему ссылку.

Чтобы перейти к пустому анализатору, дважды кликните по сообщению в журнале.

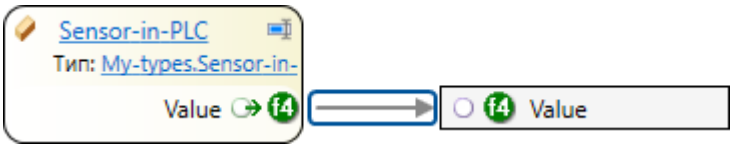
> ПРИМЕЧАНИЕ
> Чтобы быстро указать один целевой объект для нескольких инициализаторов, переключитесь в табличный вид и скопируйте целевой объект в нужные инициализаторы.

## 5.2.2.3. Добавление связей

Связь представляет собой стрелку, направленную от одного сигнала к другому.

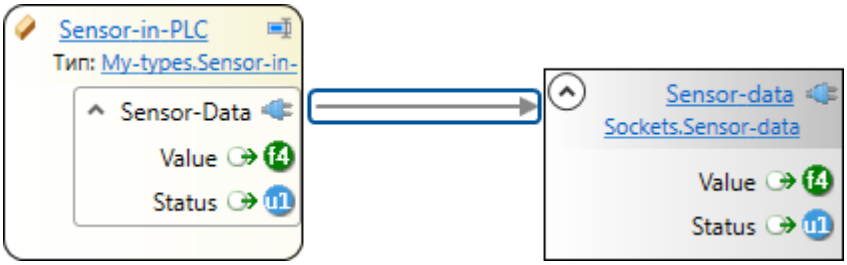

Такая стрелка означает, что при изменении значения сигнала-источника (у которого стрелка берёт начало), его новое значение будет записано в сигнал-приёмник (к которому направлена стрелка). По умолчанию, в сигнал-приёмник записывается копия значения сигнала-источника, однако в свойствах связи можно настроить другие варианты записи.

Связывать можно сигналы самого объекта, а также сигналы вложенных объектов и ссылок.




Для вложенных ссылок можно экспонировать их сигналы: при экспонировании для каждого сигнала в ссылке создаётся такой же сигнал в текущем объекте/типе и эти сигналы связываются. Чтобы выполнить экспонирование, в контекстном меню ссылки выберите **Экспонировать входы и выходы** (будут экспонированы все сигналы) или **Экспонировать отдельные сигналы...** (откроется окно выбора сигналов, которые нужно экспонировать).

Также как и сигналы, можно связывать сокеты: при связывании двух сокетов входящие в них сигналы связываются автоматически.



ОБРАТИТЕ ВНИМАНИЕ

Сигналы сокетов связываются по имени: для каждого сигнала сокета-источника в сокете-приёмнике должен быть сигнал с тем же именем. Тип значения сигнала-источника должен быть таким же, как у сигнала-приёмника или должен неявно к нему приводиться.

Чтобы добавить связь:

1. Протяните стрелку от источника (сигнала или сокета) к приёмнику (сигналу или сокету).
2. (Опционально) В свойствах связи настройте вид связи.

## Виды связи

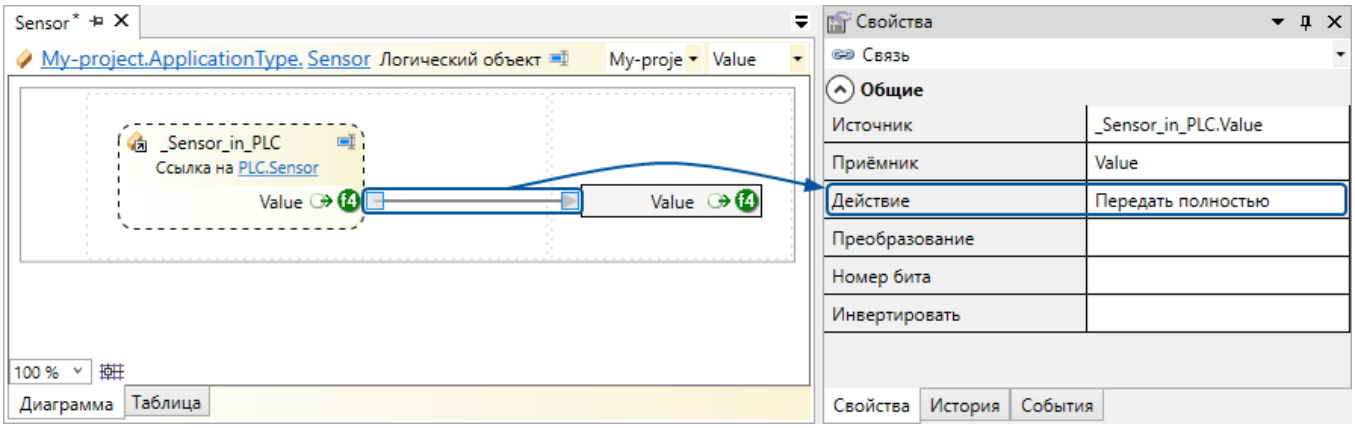
### Передать полностью

В приёмник записывается копия VQT источника.

Когда используется:

- при передаче значений и команд между компонентами.

Чтобы настроить копирование, укажите в свойствах связи **Действие** - **«Передать полностью»** (указано по умолчанию при создании связи).




### Передать значение одного бита

В приёмник записывается значение одного бита источника.

Когда используется:

- если значение, которое передаётся в SePlatform.Data Server, представляет собой набор битов, составленный из значений нескольких булевых параметров. После получения такого набора битов, нужно значение каждого бита записать в отдельный сигнал, описывающий параметр.

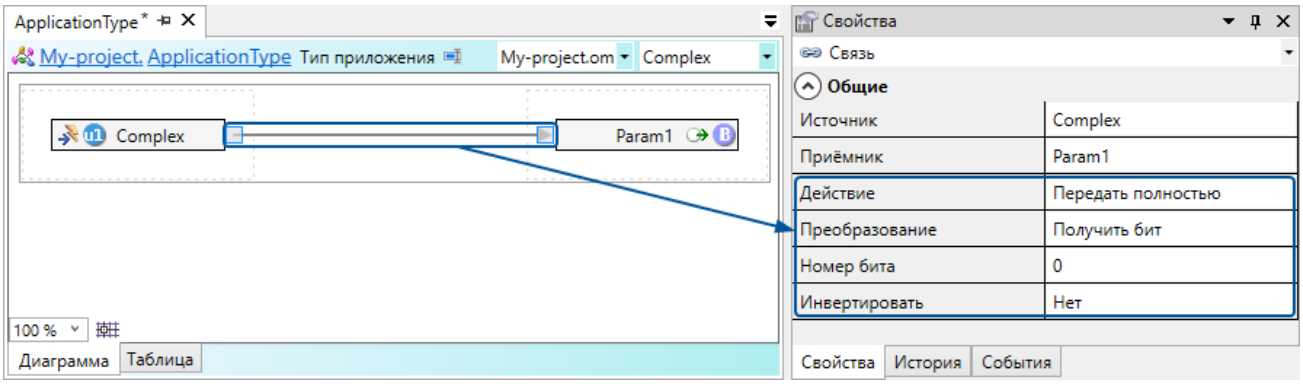
Тип источника: int1, uint1, int2, uint2, int4, uint4, float, double.

Тип приёмника: bool.

Как настроить:

1. Протяните стрелку связи от сигнала-источника, в который записывается набор битов, к сигналу-приёмнику.

2. Задайте свойства связи:
   - **Действие** - **«Передать полностью»**;
   - **Преобразование** - **«Получить бит»**;
   - **Номер бита** - номер бита источника, который нужно записать в сигнал-приёмник;
   - **Инвертировать** - укажите **«Да»**, если при записи нужно инвертировать значение бита.




Описанную настройку нужно выполнить для каждого булевого параметра, передаваемого в составе набора битов.

## Сформировать качество приёмника

Значению приёмника устанавливается качество в зависимости от значения источника.

Когда используется:

- если качество передаётся в SePlatform.Data Server отдельно от значения с помощью сигнала типа bool - достоверное/недостоверное.

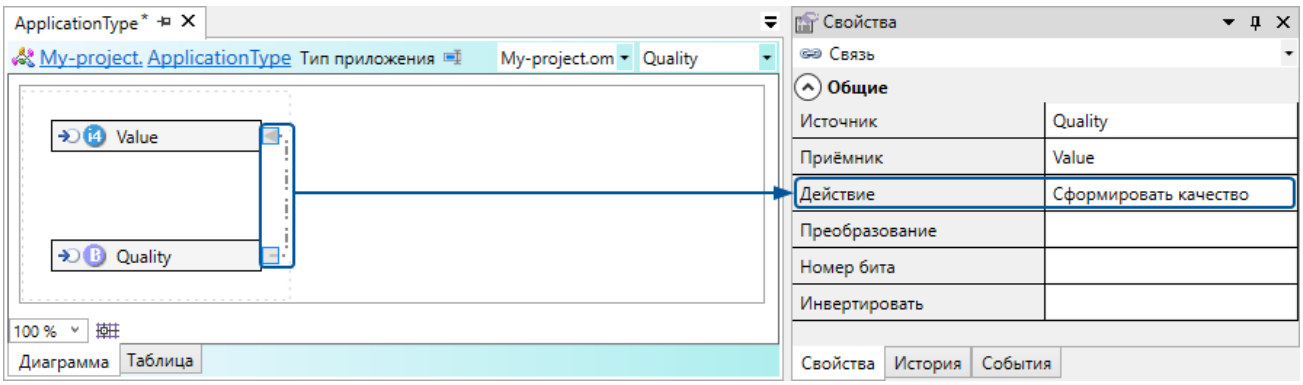
Тип источника: bool.

Тип приёмника: любой.

Как настроить:

1. У связи, по которой передаётся значение параметра, в свойстве **Действие** выберите **«Только значение»**.
2. Протяните связь от сигнала, в который записывается качество, к сигналу в который записывается значение.

   У связи в свойстве **Действие** выберите **«Сформировать качество»** или **«Сформировать качество с инверсией»**.



## Передать статус доставки команды

В приёмник записывается статус доставки команды.

Когда используется:

- при передаче управляющих команд от SePlatform.Data Server к устройствам по спецификациям/протоколам, приведённым в таблице ниже.

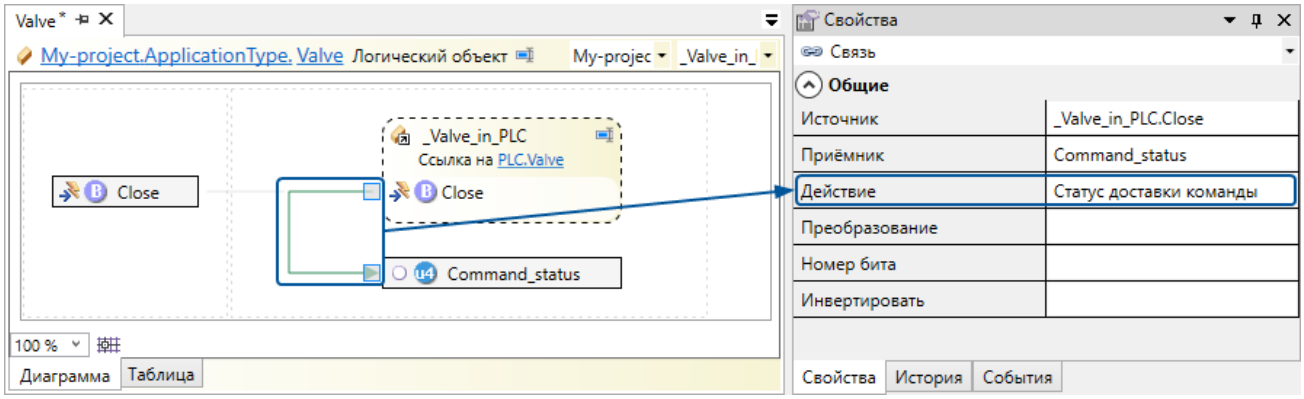
Источник - сигнал, в который передаётся команда.

Тип приёмника - зависит от протокола/спецификации, по которому передаётся команда

| Спецификация/протокол | Тип статуса доставки команды |
|---|---|
| OPC DA | uint4 |
| Modbus TCP | int4 |
| Modbus RTU | int4 |
| МЭК 870-5-101 | int4 |
| МЭК 870-5-104 | int4 |
| МЭК 61850 | int1, int2, int4, int8 |

Как настроить:

1. В объект, передающий команду, добавьте выходной сигнал нужного типа (см. таблицу).
2. Протяните связь от сигнала, в который передаётся команда, к добавленному сигналу и в свойстве **Действие** выберите **«Статус доставки команды»**.




## Передать признак обновления данных

В приёмник записывается значение **«true»**, когда данные по объекту переданы от устройства в SePlatform.Data Server.

Когда используется:

- при использовании в SePlatform.Data Server логического адаптера **Опросчик МЭК 61850**, выполняющего периодический опрос устройств.

Источник - любой сигнал объекта в устройстве, значение которого передаётся в SePlatform.Data Server при опросе.

Тип сигнала-приёмника - bool.

Пример использования данного вида связи приведён в разделе 5.13. МЭК 61850: формирование параметра из его атрибутов (стр. 124).

## Передать значение сервисного сигнала

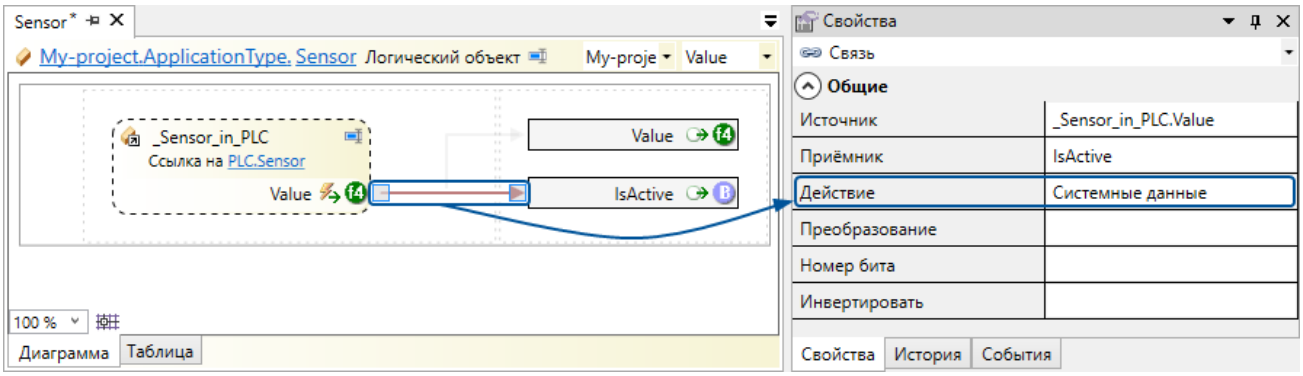
В приёмник записывается значение сервисного сигнала коммуникационного модуля SePlatform.Data Server.

Когда используется:

- при необходимости обрабатывать/устанавливать значение сервисного сигнала следующих модулей SePlatform.Data Server:
  - **OPC DA Клиент**
  - **OPC UA Клиент**
  - **Опросчик МЭК 60870-5-101**
  - **Опросчик МЭК 60870-5-104**
  - **Опросчик МЭК 61850**
  - **Опросчик Modbus TCP**
  - **Опросчик Modbus RTU**

Как настроить:

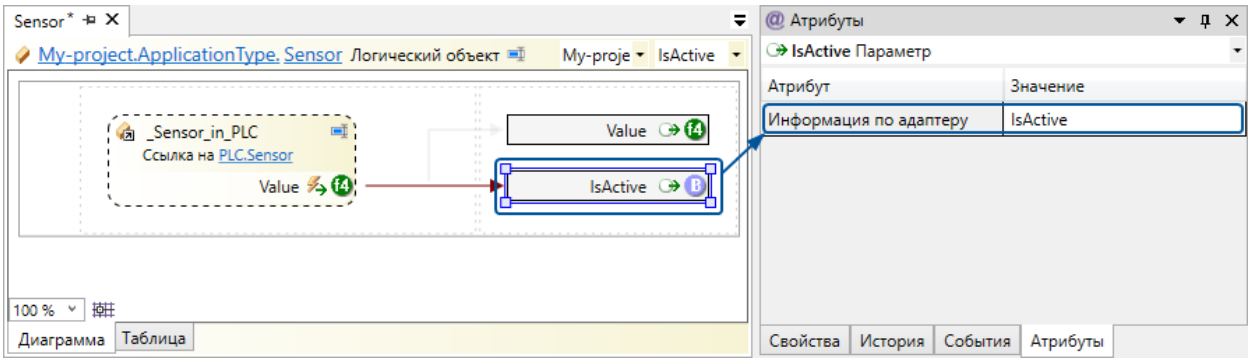
1. Добавьте в нужное место приложения для SePlatform.Data Server выходной сигнал. Тип значения устанавливается в соответствии с типом сервисного сигнала.
2. Протяните связь от сигнала удалённого объекта, связанного с сигналом в SePlatform.Data Server, к добавленному сигналу и у связи в свойстве **Действие** выберите **«Системные данные»**.



3. В атрибутах добавленного сигнала укажите сервисный сигнал:

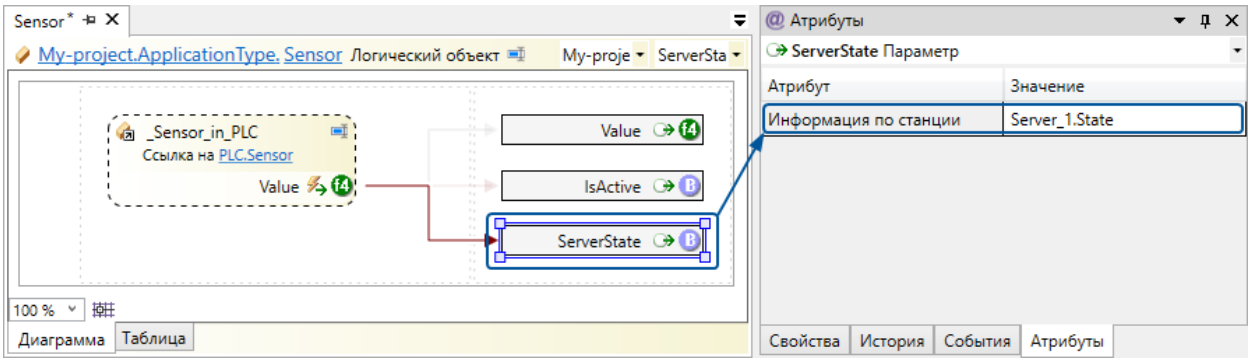
   - чтобы указать сервисный сигнал коммуникационного модуля, с помощью которого передаются данные, добавьте сигналу атрибут **Информация по адаптеру** и укажите тег сервисного сигнала в узле коммуникационного модуля.

     Атрибут расположен в узле `Server.Om → Attributes → SystemData → Communications`.




   - чтобы указать сервисный сигнал источника, от которого приходят данные или которому они передаются, добавьте сигналу атрибут **Информация по станции** и укажите тег сервисного сигнала в узле станции.

     Атрибут расположен в узле `Server.Om → Attributes → SystemData → Communications`.




## 5.2.2.4. Добавление вычислений

Вычисления представляют собой исходный код, в результате исполнения которого вычисляются значения одного или нескольких сигналов.

Виды вычислений:

- исполняемая процедура
- формула сигнала
- файл динамической библиотеки

## Исполняемая процедура

При использовании данного вида вычислений, в решение добавляется отдельный элемент, в котором описывается исходный код исполняемой процедуры. Описание вычислений в исполняемой процедуре позволяет:

- добавлять исходный код с использованием любых конструкций языка SePlatform.Om.
- настраивать условия исполнения процедуры:
  - при изменении значений сигналов
  - по таймеру
  - по расписанию

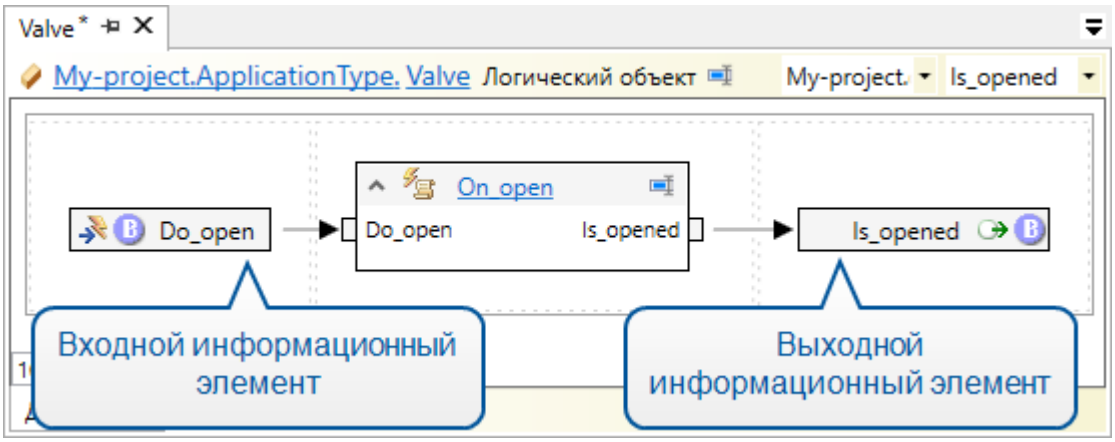
Исполняемой процедуре можно назначить любое количество условий исполнения.

Чтобы добавить исполняемую процедуру:

1. В объект/тип добавьте элемент **Обработчик события**.
2. Добавьте исходный код исполняемой процедуры:
   - в редакторе исходного кода: открывается при переходе в элемент **Обработчик события**
   - в свойстве **Код обработки**
   - в окне **Формулы**

После добавления исходного кода, на элементе отобразятся входные и выходные сигналы этой процедуры:

- входные сигналы - сигналы, значения которых используются в вычислениях.
- выходные сигналы - сигналы, значения которых вычисляются при исполнении процедуры.



3. Добавьте условия исполнения процедуры:
   - При изменении значений сигнала:
     1. Свяжите сигнал с элементом **Обработчик события**.
     2. В свойствах связи укажите условия, при которых должен исполняться код:
        - **«Изменение значения»** - при изменении значения или качества сигнала.
        - **«Обновление значения»** - даже при полном повторе значения, качества и метки времени сигнала.
        - **«Подготовка сообщения»** - при изменении значения, качества или метки времени сигнала. Код будет выполнен до генерации события.

     > ОБРАТИТЕ ВНИМАНИЕ
     > Процедура будет исполняться при изменении значений только тех сигналов, для которых настроена связь с **Обработчиком события**.

   - По таймеру:
     1. Добавьте элемент **Таймер**.
     2. В свойствах укажите период срабатывания таймера в миллисекундах.
     3. Свяжите **Таймер** и **Обработчик события**.

- По расписанию:
  1. Добавьте элемент **Расписание**.
  2. В свойствах укажите cron-выражение, определяющее периодичность исполнения кода.
  3. Свяжите **Расписание** и **Обработчик события**.

Cron-выражение - строка, состоящая из 6 полей, разделённых пробелами:

| Поле | Значения |
|---|---|
| Секунды | 0 - 59 |
| Минуты | 0 - 59 |
| Часы | 0 - 23 |
| Число месяца | 1 - 31 |
| Месяц | JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC<br>1 - 12 |
| День недели | SUN, MON, TUE, WED, THU, FRI, SAT<br>0 - 7 (значения 0 и 7 соответствуют воскресенью) |

В каждом поле можно указать:

- Одно значение - условие будет выполняться только для указанного в поле значения. Например, если в поле Часы указать **«12»**, то условие будет выполняться в 12 часов.
- **«*»** - условие будет выполняться для каждого значения. Например, если в поле Месяц указать **«*»**, то условие будет выполняться в любой месяц.
- Несколько значений через запятую - условие будет выполняться для каждого из значений. Например, если в поле День недели указать **«SAT,SUN»**, то условие будет выполняться в субботу и воскресенье.
- Диапазон значений через дефис - условие будет выполняться для каждого значения из диапазона. Например, если в поле День недели указать **«MON-FRI»**, то условие будет выполняться с понедельника по пятницу.
- Диапазон значений с интервалом (интервал указывается через **«/»** после диапазона) - условие будет выполняться для значений в указанном диапазоне с заданным интервалом. Например, если в поле Число месяца указать 1-20/5, то условие будет выполняться с 1 по 20 числа месяца каждые 5 дней: 1, 6, 11, 16. Вместо диапазона можно указать **«*»**: в этом случае условие будет выполнено для всего диапазона значений. Например, если в поле Часы указать **«*/3»**, то условие будет выполнено каждые 3 часа
- **«?»** - может быть указан в поле Число месяца или День недели и только в одном из них. Означает, что поле не используется в выражении, вместо него используется значение, указанное во втором поле. Например, чтобы условие выполнялось каждое 10 число месяца и неважно, какой это будет день недели, нужно в поле Число месяца указать **«10»**, а в поле День недели - **«?»**

ПРИМЕР

Каждый день в 12:00:

```
0 0 12 * * ?
```

Каждый день в 23:59:59:

```
59 59 23 ? * *
```

Каждую минуту с 0:00 до 0:59 каждый день:

```
0 * 0 * * ?
```

В 12:00 в субботу и воскресенье:

```
0 0 12 ? * 6,7
```

В 0:00 с понедельника по пятницу:

```
0 0 0 ? * MON-FRI
```

Раз в час с 0:00 до 6:00:

```
0 0 0-6 * * ?
```

В воскресенье каждые 4 часа:

```
0 0 */4 ? * SUN
```

Каждый день с 3:00 до 3:30 каждые 2 минуты:

```
0 0-30/2 3 * * ?
```

Каждый год 31 декабря в 23:59:59:

```
59 59 23 31 DEC ?
```

## Формула сигнала

Формулу следует использовать, если значение сигнала является результатом обработки одного или нескольких сигналов и должно пересчитываться при каждом изменении любого из них.

ОБРАТИТЕ ВНИМАНИЕ

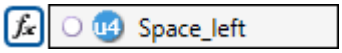
Формулу можно использовать только для следующих типов сигналов:

- переменная;
- выходной параметр (параметр, у которого в свойстве **Направление** указано значение **«Выход»**).

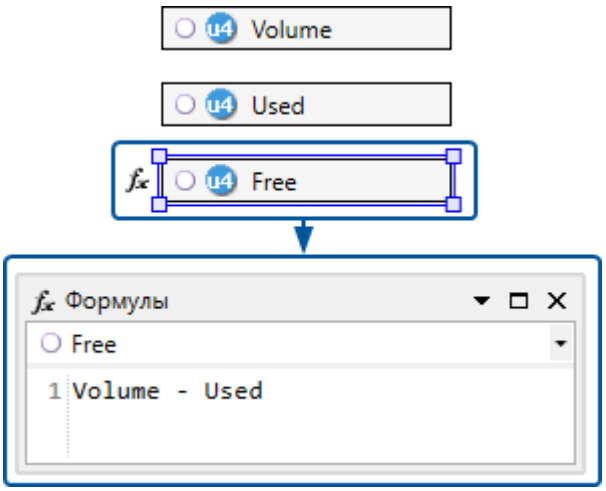
Чтобы добавить формулу:

1. Выберите сигнал, значение которого должно вычисляться.
2. В окне **Формулы** задайте формулу, по которой должно вычисляться значение элемента:
   - формула должна быть написана на языке SePlatform.Om.
   - формула должна описывать выражение, результат вычисления которого будет присвоен сигналу в качестве значения.
   - результат вычисления должен быть того же типа, что и тип значения сигнала, или неявно к нему приводиться.
   - формула должна содержать хотя бы один сигнал без оператора read[1]: в противном случае формула не будет запущена никогда.
   - в формуле можно использовать значение самого сигнала, значение которого вычисляется: формула не будет пересчитываться при изменении значения этого сигнала.

После задания формулы, около информационного элемента появится иконка рассчёта значения.




ПРИМЕР




## Файл динамической библиотеки

Данный способ описания вычислений предназначен для добавления в решение процедур и функций, реализованных в файлах динамических библиотек.
Чтобы использовать в решении функцию из динамической библиотеки, необходимо добавить в решение элемент внешней функции и настроить его. После настройки элемента, внешнюю функцию, можно будет вызывать в исполняемых процедурах и формулах, написанных на языке SePlatform.Om.
Чтобы добавить в решение функцию, описанную в файле динамической библиотеки, выполните следующие действия:

1. Добавьте в решение элемент **Внешняя функция**.
2. В свойстве **Имя файла** укажите путь к файлу динамической библиотеки.
3. В свойстве **Имя функции** укажите имя функции, описанной в динамической библиотеке.

   Если файл динамической библиотеки содержит несколько функций, необходимо добавить в решение соответствующее количество элементов и выполнить описанную настройку для каждого из них.

---

[1]Если в формуле указать read перед именем сигнала, она не будет пересчитываться при изменениях значений этого сигнала.

4. В свойстве **Тип возвращаемого значения** укажите тип значения, возвращаемого указанной функцией.
5. На образе элемента добавьте параметры функции.

Для каждого параметра функции настройте его свойства:

- **Тип значения** - тип данных параметра функции;
- **Направление** - направление передачи параметра;
- **Имя** - название параметра.

## 5.2.2.5. Настройка генерации событий

Генерация событий настраивается для объектов, размещаемых в SePlatform.Data Server.

Чтобы настроить генерацию событий:

1. Выберите сигнал, при изменении значений которого будут генерироваться события.

2. В окне **События**:

2.1. Установите флаг **Генерировать события**.

2.2. Выберите тип условия. Доступные типы условий зависят от типа сигнала.

| Тип сигнала | Типы условий |
|---|---|
| bool | › **Дискретный** (не требуется выбирать) |
| int1, uint1,<br>int2, uint2,<br>int4, uint4,<br>int8, uint8 | › **Перечисление**<br>› **По уровню**<br>› **Отклонение** |
| float, double | › **По уровню**<br>› **Отклонение** |
| string | › **Динамическое** (не требуется выбирать) |

ПРИМЕЧАНИЕ

Подробнее о типах условий сказано в документации на модуль OPC AE Server.

2.3. Укажите параметры подусловий события.

| Параметр | Описание |
|---|---|
| Столбец без заголовка | Включить/отключить генерацию событий при выполнении подусловия. |
| **Подусловие** | Название подусловия. |
| **Значение** | Значение, используемое в подусловии. У нечисловых типов (bool, string) отсутствует. |
| **Сообщение** | Текст сообщения при выполнении подусловия. При генерации, конкатенируется с именем события: **«<имя объекта><сообщение>»**.<br>Для удобочитаемости, рекомендуем в начало сообщения добавлять пробел или точку с пробелом. |
| **Важность** | Важность события при выполнении подусловия |
| **Квитировать** | Требование квитировать событие:<br>› **«true»** - требует квитирования<br>› **«false»** - не требует квитирования |
| **Звук** | Имя звукового файла, который будет проигрываться при генерации события. Можно указать несколько файлов через запятую: файлы будут воспроизводиться последовательно один за другим. Расширение файлов указывать необязательно. |

Чтобы добавить подусловие, в контекстном меню выберите **Добавить подусловие** или нажмите клавишу **«Insert»**.

После настройки генерации событий у сигнала появится атрибут **События**, значение которого будет содержать настроенные параметры генерации.

ПРИМЕЧАНИЕ

Значение атрибута можно редактировать в окне **Атрибуты** (стр. 167). Например, можно настроить вычисление значений для отдельных полей атрибута.

3. У объекта, в котором расположен сигнал, в окне **События** установите флаг **Генерировать события** (по умолчанию установлен).

Если флаг не установлен, события в объекте генерироваться не будут (но могут генерироваться во вложенных объектах).

Если сигнал описан в типе, то флаг **Генерировать события** устанавливается для каждого объекта этого типа независимо.

ОБРАТИТЕ ВНИМАНИЕ

Чтобы SePlatform.Data Server генерировал события, нужно добавить модуль OPC AE Server в его описание (стр. 39).

ПРИМЕЧАНИЕ

События можно сохранять в историю (стр. 94).

## 5.2.3. Размещение объектов

После того, как описаны типы, можно размещать объекты описанных типов в приложениях. При размещении сначала создаются объекты в одном аспекте, а после этого создаются их представления в других аспектах (в том же приложении или в другом).

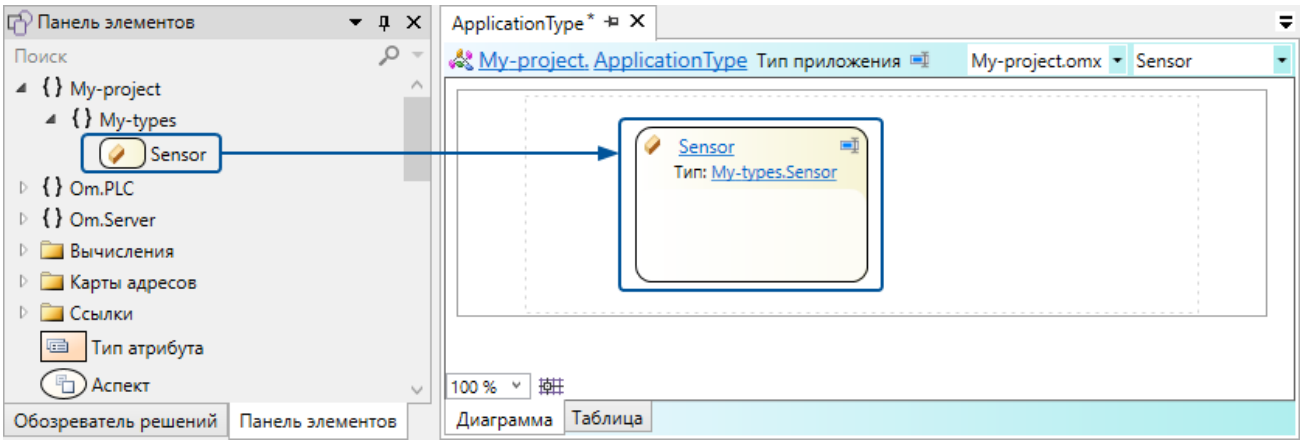
ПРИМЕЧАНИЕ

В разделе описано, как размещать объекты в приложениях и их типах. Таким же образом можно размещать объекты в объектах и их типах.

### Создание объектов

1. Перейдите в **Приложение** или **Тип приложения**.
2. Добавьте объекты нужных типов:
   2.1. В панели элементов раскройте узел с именем проекта.
   2.2. Найдите в узле объект нужного типа. Объект находится по тому же пути, что и тип в проекте.
   2.3. Перетащите элемент объекта в рабочую область.

   Будет создан объект выбранного типа, объект будет иметь тот же аспект, что и тип.




После создания объектов, нужно создать их представления в других аспектах.

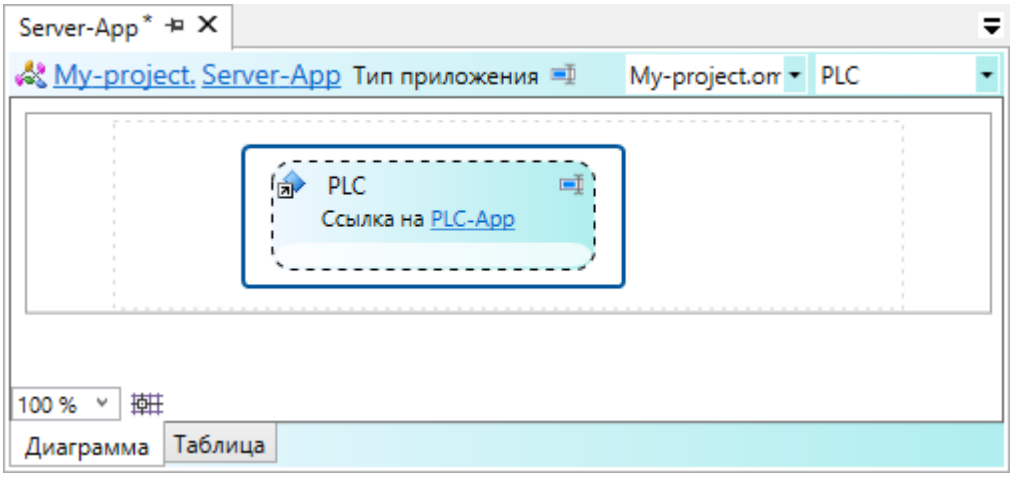
### Создание представлений

Для быстрого создания представлений используются мастеры.

| Мастер | Описание | Когда использовать |
|---|---|---|
| Мастер создания представлений | Размещает в приложении представления нескольких объектов | При добавлении приложения |
| Мастер отображения объектов | Размещает представления одного объекта в нескольких приложениях | При добавлении объекта в существующее решение |

## Мастер создания представлений

1. Перейдите в **Приложение** или **Тип приложения**.
2. Если объекты описаны в типе приложения, добавьте ссылку на него:
   2.1. Добавьте элемент **Ссылка**.
   2.2. В свойстве **Тип** укажите тип приложения.




ОБРАТИТЕ ВНИМАНИЕ

При добавлении приложения в домен ссылку нужно будет инициализировать: указать конкретное приложение в домене, на которое она ссылается (стр. 67).

3. Создайте представления объектов.

Для этого в контекстном меню выберите **Создать здесь представления объектов...**: откроется мастер создания представлений. Далее следуйте указаниям мастера.

В результате:

- Для каждого выбранного объекта будет создано его представление (объект) в указанном аспекте.
- Каждое представление будет иметь тип, описанный для выбранного аспекта (если при описании типов для них были указаны представляемый тип и аспект).
- Будут инициализированы аспектные ссылки, добавленные в типы.

ПРИМЕР

В проекте описаны типы для задвижек и датчиков давления:

- в аспекте **«ПЛК»** описаны типы **«Задвижка»** и **«Датчик»**, в них – только сигналы.
- в аспекте **«Сервер»** описаны типы **«Задвижка»** и **«Датчик»**, в каждом – аспектная ссылка (ссылка, указывающая на соответствующий тип в аспекте **«ПЛК»**) и экспонированные сигналы.

Нужно описать приложения для ПЛК и для SePlatform.Data Server.

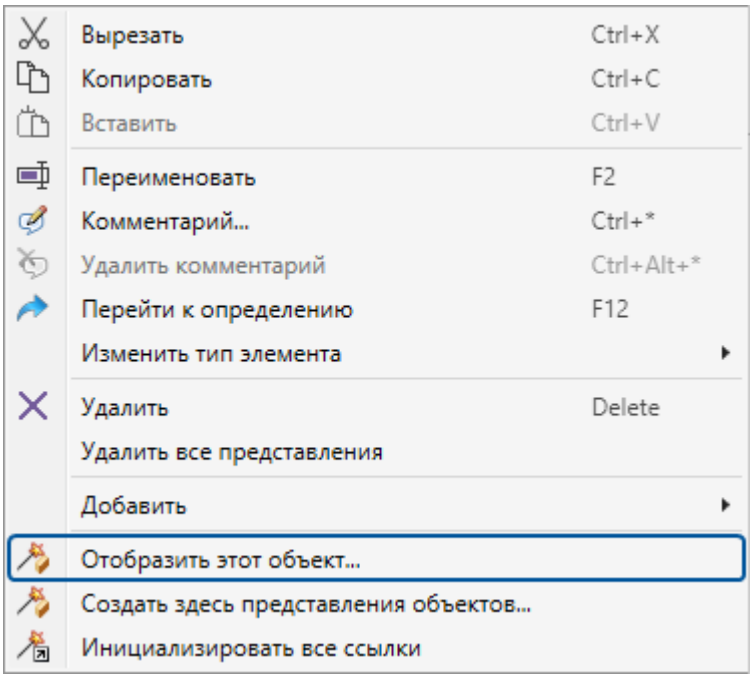
1. Создаём тип приложения для ПЛК: **«PLC_App»**.
2. В него добавляем один объект типа **«Задвижка»** и два объекта типа **«Датчик»**: **«Давление до»** и **«Давление после»**.
3. Создаём тип приложения для SePlatform.Data Server: **«Server_App»**.
4. В него добавляем ссылку на **«PLC_App»**.
5. Создаём представления:
   5.1. Вызываем мастер.
   5.2. Добавляем в список объекты, расположенные в **«PLC_App»**.
   5.3. Выбираем аспект – **«Сервер»**.
   5.4. Нажимаем **Ок**.

В результате:

- В **«Server_App»** будут созданы объекты: **«Задвижка»**, **«Датчик до»** и **«Датчик После»**.
- Для каждого объекта указаны:
  - представляемый объект – соответствующий объект в **«PLC_App»**.
  - аспект – **«Сервер»**.
  - тип – соответствующий тип в аспекте **«Сервер»**.
- В каждом объекте аспектная ссылка инициализирована соответствующим объектом в **«PLC_App»**.

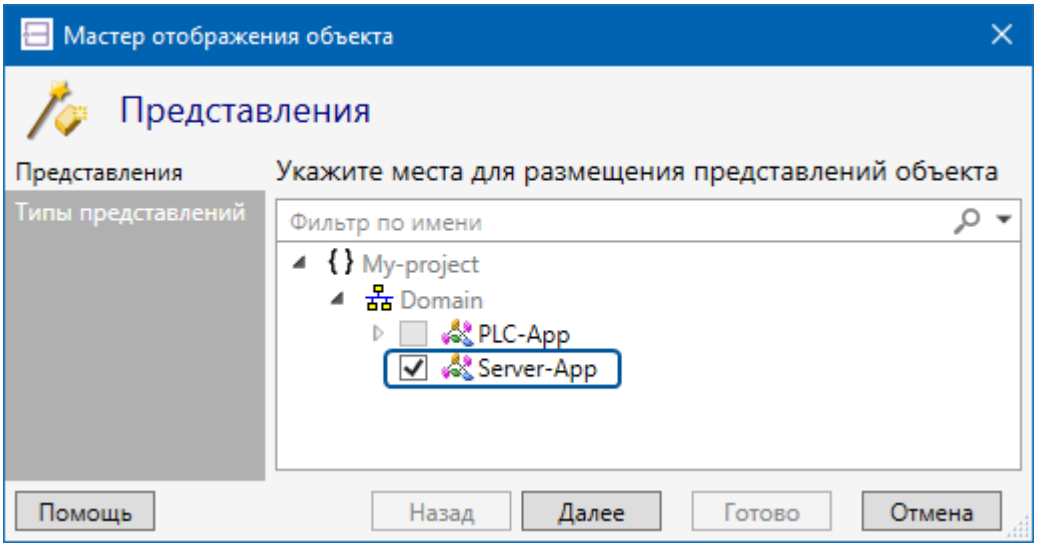
## Мастер отображения объектов

1. Перейдите к объекту, для которого нужно создать представления.
2. В контекстном меню объекта выберите **Отобразить этот объект...**.

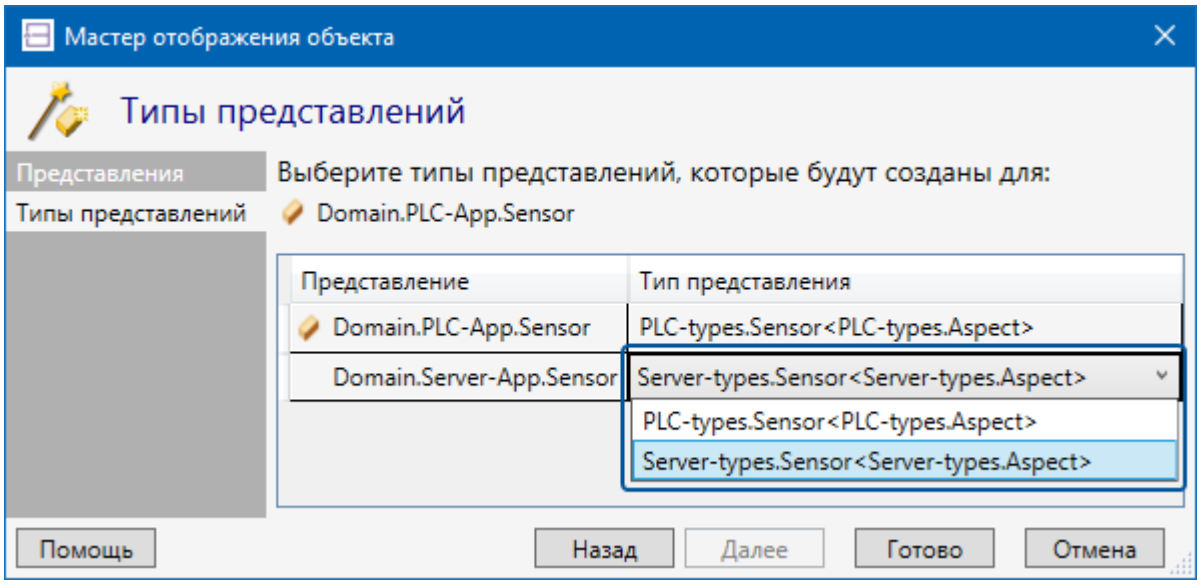



Откроется окно мастера.

3. Выберите места, куда нужно добавить представления и нажмите **Далее**.



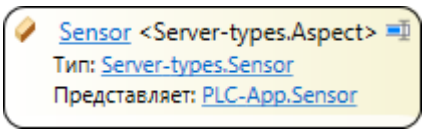
4. Для каждого добавляемого представления укажите тип.




5. Нажмите **Готово**.

В результате:

- В каждом выбранном месте будет создано представление объекта.
- Каждое представление будет иметь указанный ему тип.
- У каждого представления будут инициализированы аспектные ссылки, добавленные в тип.




ОБРАТИТЕ ВНИМАНИЕ

Ссылки не будут инициализированы, если объекты находятся в разных адресных пространствах (например, в разных типах приложений). В этом случае необходимо инициализировать ссылки вручную.

ПРИМЕР

В проекте описаны типы для датчика температуры:

- в аспекте **«ПЛК»** описан тип **«Датчик»**, в нём сигналы: измеренная температура и статистика опроса датчика контроллером - количество неуспешных запросов.
- в аспекте **«Сервер»** описан тип **«Датчик»**, в нём аспектная ссылка на **«Датчик»** в аспекте **«ПЛК»** и сигнал, в который записывается полученное значение температуры.
- в аспекте **«Статистика»** описан тип **«Датчик»**, в нём аспектная ссылка на **«Датчик»** в аспекте **«ПЛК»** и сигнал, в который записывается количество неуспешных запросов.

Нужно описать приложения для ПЛК и для двух экземпляров SePlatform.Data Server: один будет принимать данные датчика, другой - статистику.

1. Создаём приложения для ПЛК и двух экземпляров SePlatform.Data Server: **«PLC_App»**, **«Server_App»**, **«Statistics_Server_App»**.
2. В **«PLC_App»** добавляем объект типа **«Датчик»** в аспекте **«ПЛК»**.
3. Создаём представления:
   3.1. В контекстном меню объекта вызываем мастер.
   3.2. Выбираем, куда нужно добавить представления: приложения **«Server_App»** и **«Statistics_Server_App»**.
   3.3. Для каждого представления указываем тип.
   3.4. Нажимаем **«Готово»**.

В результате:

- В приложении **«Server_App»** будет создан объект **«Датчик»**, для него указаны:
  - представляемый объект - **«Датчик»** в приложении **«PLC_App»**.
  - аспект - **«Сервер»**.
  - тип - **«Датчик»** в аспекте **«Сервер»**.
- В приложении **«Statistics_Server_App»** будет создан объект **«Датчик»**, для него указаны:
  - представляемый объект - **«Датчик»** в приложении **«PLC_App»**.
  - аспект - **«Статистика»**.
  - тип - **«Датчик»** в аспекте **«Статистика»**.
- Аспектные ссылки в обоих объектах будут инициализированы объектом **«Датчик»** в приложении **«PLC_App»**.

# 5.3. Совместная разработка проекта несколькими пользователями

SePlatform.Development Studio предоставляет возможность совместной разработки одного проекта несколькими пользователями.

Для совместной разработки применяется система управления версиями SVN. Прежде чем перейти к внедрению SVN, можно ознакомиться с документацией для этой системы: https://svnbook.red-bean.com/.

Чтобы приступить к совместной разработке проекта, выполните описанные ниже шаги.

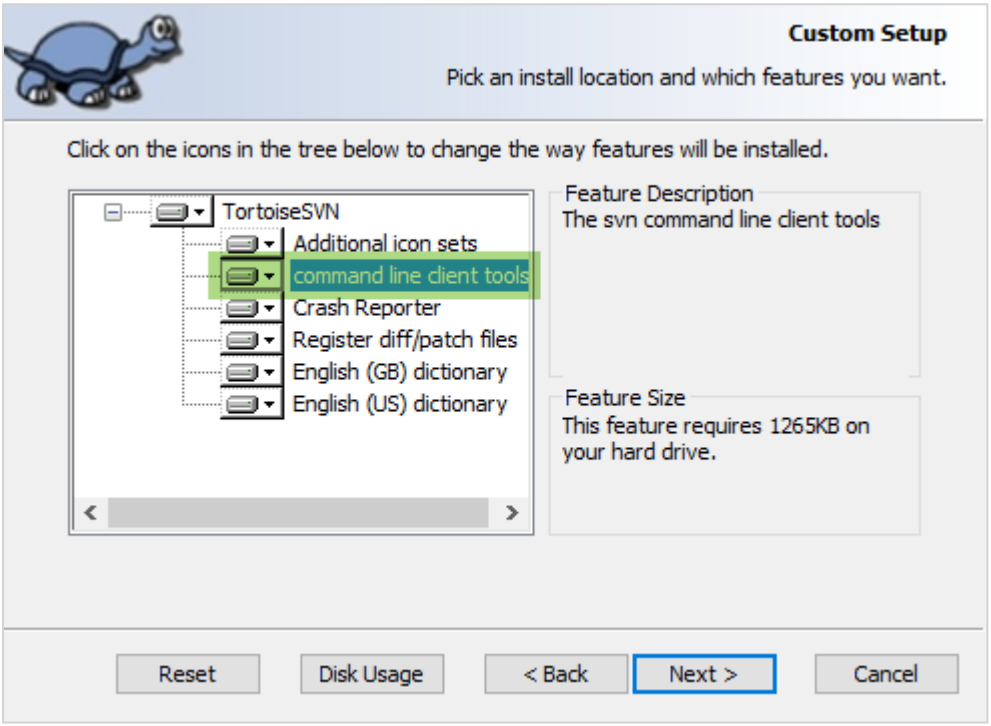
1. Создайте репозиторий на SVN сервере для хранения файлов проекта.

   Можно использовать имеющийся или развернуть собственный SVN сервер. Для создания собственного сервера можно использовать, например, программу VisualSVN Server.

Затем следует создать на SVN сервере репозиторий. Структура репозитория может быть любой. Для примера примем, что файлы проекта будут храниться в папке `project` репозитория `test`: `http://test.local/svn/test/project`.

Также в репозитории необходимо создать папку для хранения внутренних идентификаторов объектов (сигналов, приложений и пр.) - `sysevo`. Внутренние идентификаторы используются для обращения к определенным сигналам, например, при запросе исторических значений. При построении проекта любым из пользователей должны использоваться одни и те же идентификаторы. Для примера примем, что такая папка находится по адресу `http://test.local/svn/test/sysevo`.

2. На каждом компьютере, где будет вестись разработка проекта, установите клиентское приложение SVN - TortoiseSVN. Обратите внимание, что при установке обязательно нужно отметить компонент **`command line client tools`**.




3. Разместите файлы проекта в предназначенной для этого папке репозитория.

Теперь можно приступить к совместной разработке проекта.

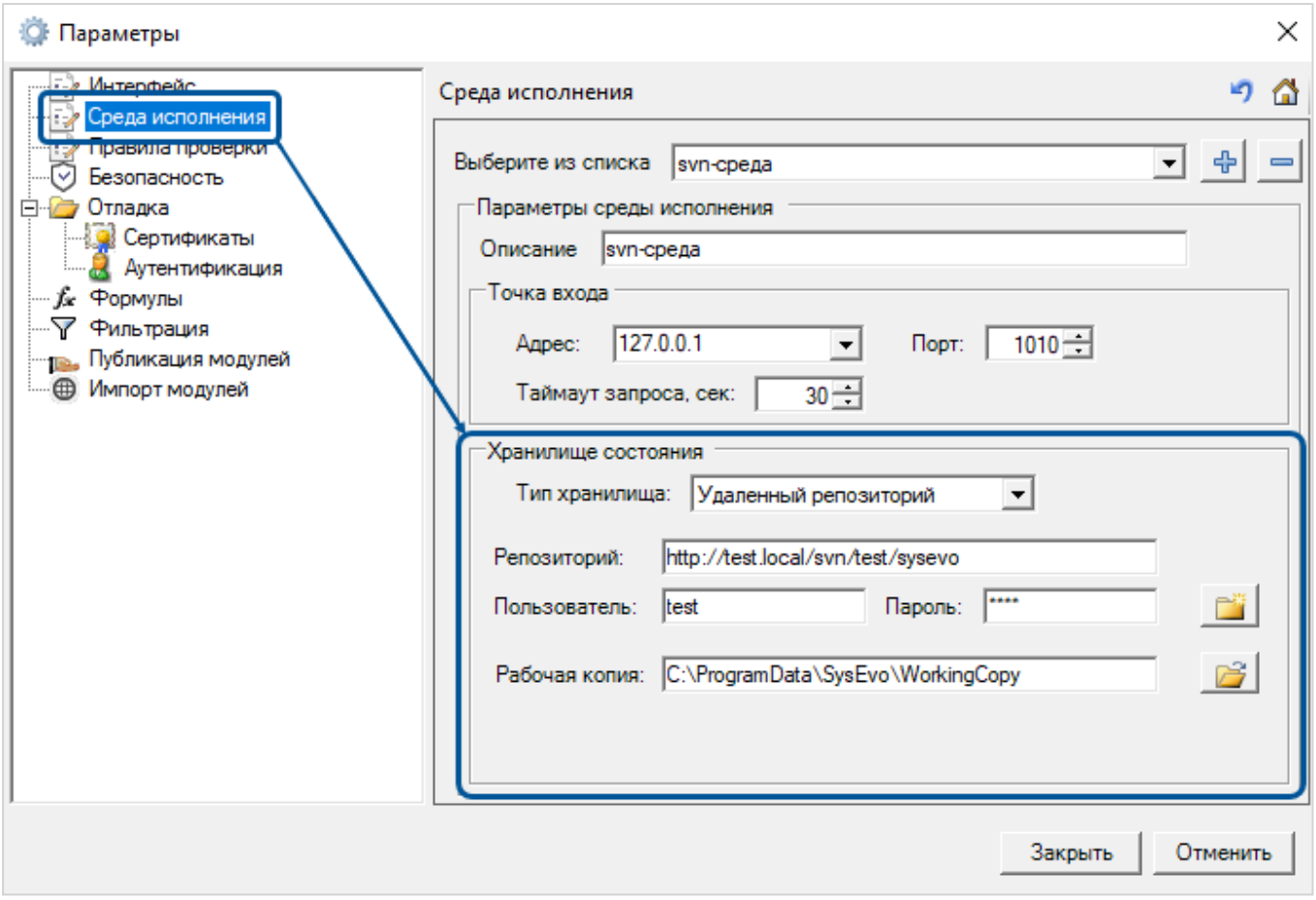
## Извлечение и работа с локальной копией проекта

Чтобы приступить к работе с проектом, необходимо с помощью TortoiseSVN извлечь копию проекта из репозитория в локальную папку того компьютера, на котором будет вестись разработка.

Затем в настройках локальной копии проекта необходимо указать параметры хранилища состояний. Под хранилищем состояний подразумевается папка для хранения внутренних идентификаторов объектов. Перейдите в меню **Файл** → **Параметры...** → **Среда исполнения**. Укажите в разделе **Хранилище состояния** следующее:

- тип хранилища - **«Удаленный репозиторий»**;
- репозиторий - адрес папки для хранения внутренних идентификаторов объектов (в примере, описанном выше - `http://test.local/svn/test/sysevo`);
- учетные данные для доступа к репозиторию.

В поле **Рабочая копия** по умолчанию указан адрес локальной папки для хранения идентификаторов. Обратите внимание, что папка должна быть доступна для записи.



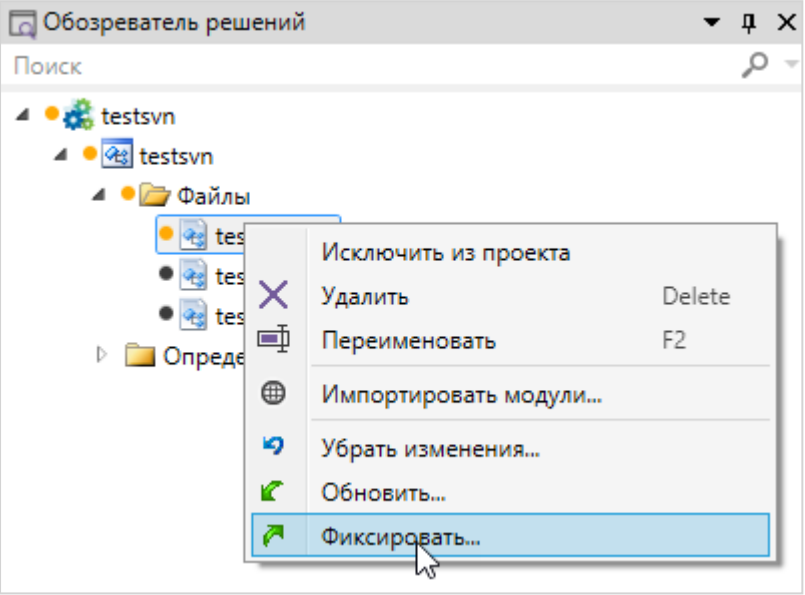

Теперь можно приступить к работе с проектом.

Использование системы контроля версий позволяет хранить разные состояния проекта. Так, в репозитории хранится одна версия проекта, а в локальной копии - другая. Чтобы изменения, сделанные локально, появились в репозитории, их необходимо туда отправить. Если файлы проекта извлечены из репозитория SVN, то в обозревателе решений появляются:

- Команды в контекстном меню элементов файлов, папок проектов и решения:
  - **Убрать изменения** - отменяет все локальные изменения файла;
  - **Обновить** - подгружает изменения файла, отправленные в репозиторий из других локальных копий;
  - **Фиксировать** - отправляет изменения файла в репозиторий.
- Иконки, свидетельствующие о состоянии файлов, где:
  - зеленый цвет иконки означает, что локальная копия файла не была изменена с момента ее получения с сервера;
  - желтый цвет иконки означает, что файл был изменен локально, и эти изменения не были отправлены на сервер;
  - красный цвет иконки означает, что файл находится в конфликтном состоянии. Оно возникает, когда несколько разработчиков вносят изменения в одни и те же места одного файла.

ПРИМЕР

Предположим, два пользователя работают над одним и тем же проектом, внося изменения в свои локальные копии. Первый пользователь создает новый файл в проекте. Сразу после создания рядом с новым файлом (и всеми родительским объектами) появится желтая иконка. Чтобы отправить новый файл в репозиторий, первому пользователю необходимо вызвать команду **Фиксировать** в контекстном меню файла. Откроется окно, где следует описать внесенное изменение.

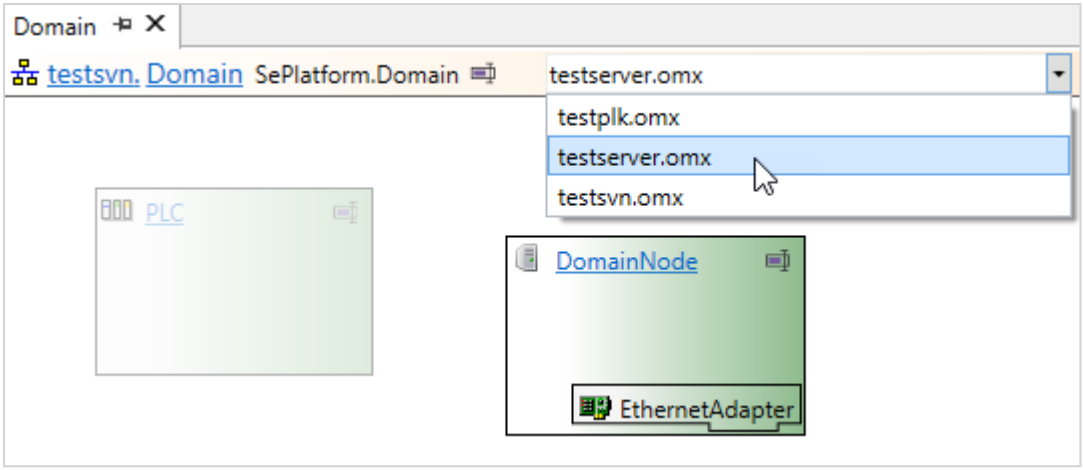



Команду можно вызвать для любого родительского объекта файла. Тогда в репозиторий можно будет отправить изменения всех файлов, вложенных в выбранный объект.

Для того, чтобы второй пользователь увидел новый файл в своей локальной копии, ему необходимо вызвать команду **Обновить** в контекстном меню решения. После этого новый файл появится в обозревателе решений.

## Рекомендации по работе с локальными копиями проекта

Рекомендуется организовать структуру проекта так, чтобы разные пользователи могли работать с разными файлами. Разбиение проекта на файлы позволяет минимизировать возможные конфликты, возникающие при редактировании одних и тех же файлов разными пользователями. Например, можно описывать ПЛК в одном файле, сервер - в другом, и так далее. На рисунке ниже ПЛК описан в файле `testplk.omx`, а SePlatform.Data Server - в файле `testserver.omx`. При открытии файла `testserver.omx` экземпляры элементов, добавленные в файл `testplk.omx`, становятся полупрозрачными, но также доступны для редактирования.


Domain
testsvn. Domain SePlatform.Domain
testserver.omx
testplk.omx
testserver.omx
testsvn.omx
PLC
DomainNode
EthernetAdapter

# 5.4. Сохранение значений и событий

SePlatform.Data Server может:

- Сохранять значения сигналов и события в историю.
- Предоставлять сохранённые данные клиентам по запросу.

Сохранённые (исторические) данные позволяют восстановить графики сигналов и возникшие события за определённый период времени. Это используется при поиске ошибок и причин неполадок, а также при анализе хода технологического процесса.

Чтобы данные сохранялись, нужно:

1. Настроить сигналам сохранение в историю. (стр. 95)

> ПРИМЕЧАНИЕ
> События настраивать не нужно: все события сохраняются в историю.

2. Добавить в домен сервер(ы) истории для хранения данных. (стр. 96)
3. Настроить сохранение данных в серверы истории. (стр. 97)
4. Настроить предоставление сохранённых данных клиентам. (стр. 98)

## Настройка сигналов

ПРИМЕЧАНИЕ

Настройки истории не влияют на сигналы, которые находятся не в SePlatform.Data Server.

Для каждого сигнала, значения которого нужно сохранять:

1. В окне **История** установите флаг **Сохранять историю**.
2. (Опционально) Установите значения фильтров:

| Фильтр | Описание |
|---|---|
| **Зона нечувствительности по значению** | Значение не будет сохранено, если оно отличается от предыдущего сохранённого меньше, чем на значение фильтра. |
| **Зона нечувствительности по времени, мс** | Интервал времени после сохранения значения, в течение которого новые значения сохраняться не будут. |

ОБРАТИТЕ ВНИМАНИЕ

Значение не будет сохранено в историю, если оно отсеяно хотя бы одним фильтром.

ПРИМЕЧАНИЕ

Фильтры используются для того, чтобы уменьшить количество сохраняемых значений. Они позволяют не записывать в историю значения сигналов, которые изменяются слишком часто или незначительно.

ПРИМЕР

Пусть датчик температуры отправляет текущую температуру раз в секунду. Чтобы не сохранять каждое полученное значение, установите в фильтре **Зона нечувствительности по значению** значение **«1»**. Теперь сохраняться будут только те значения, когда температура увеличилась или уменьшилась на 1 градус и более от предыдущего сохранённого значения.

3. (Опционально) Установите флаг **Сохранять с меткой времени сервера**.

Если флаг установлен, вместо метки времени, полученной от устройства, в историю будет сохраняться время SePlatform.Data Server в момент, когда значение было записано в сигнал.

ПРИМЕЧАНИЕ

Это нужно, если значения приходят в SePlatform.Data Server с запозданием, например из-за задержек связи. В этом случае нужно установить флаг, чтобы в историю сохранилась не метка времени, когда значение возникло, а метка времени, когда значение было получено и обработано.

ПРИМЕЧАНИЕ

Параметры, указанные в окне **История**, переносятся в атрибут сигнала **Ведение истории** (атрибут добавляется автоматически).

## Добавление серверов истории

Сервер истории - это компонент, в котором хранятся сохранённые данные. Серверы истории добавляются в домен (стр. 39). Можно добавить серверы истории следующих типов:

- SePlatform.Historian
- Microsoft SQL Server
- PostgreSQL

ОБРАТИТЕ ВНИМАНИЕ

События можно сохранять/запрашивать только из SePlatform.Historian.

ПРИМЕЧАНИЕ

В домен можно добавить несколько серверов истории. Дополнительные серверы истории можно использовать для резервирования сохраняемых данных (стр. 135).

Чтобы добавить сервер истории:

1. Если сервер истории расположен на компьютере, который уже описан в домене (элементы **Узел SePlatform.Domain** и **Компьютер**), перейдите в него.

   Если нет, добавьте в домен компьютер. Тип элемента зависит от типа сервера истории:

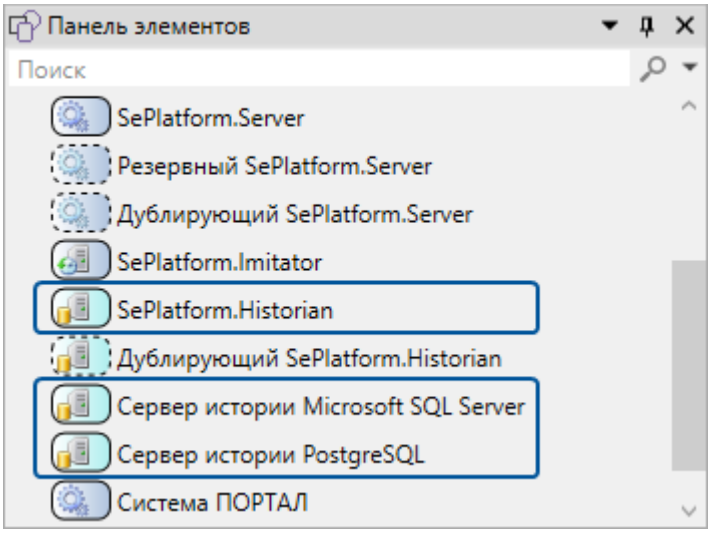
   - **Компьютер** - для Microsoft SQL Server или PostgreSQL.
   - **Узел SePlatform.Domain** - для SePlatform.Historian.

   ПРИМЕЧАНИЕ

   В **Узел SePlatform.Domain** можно добавлять серверы истории всех типов. Это нужно, например, если сервер истории расположен на компьютере, где есть SePlatform.Data Server.

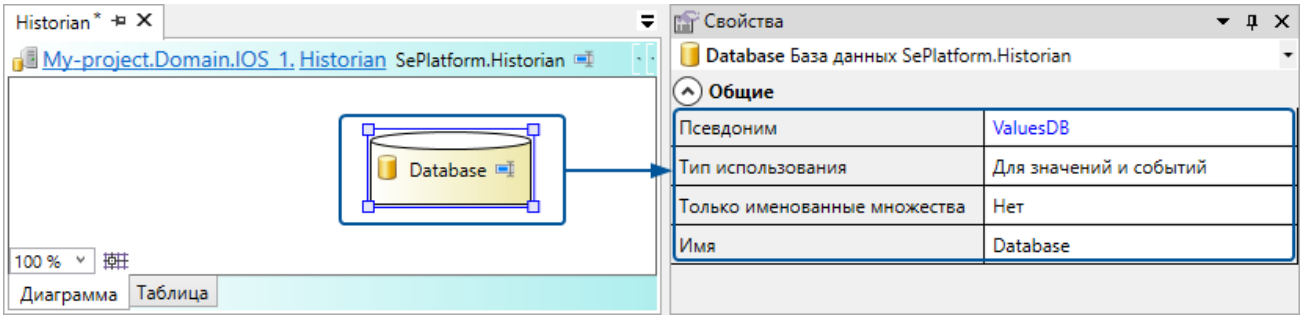
   Имя элемента - сетевое имя компьютера.

2. Добавьте сервер истории нужного типа:
   - **SePlatform.Historian**
   - **Сервер истории Microsoft SQL Server**
   - **Сервер истории PostgreSQL**




> ОБРАТИТЕ ВНИМАНИЕ
> На одном компьютере может быть только один сервер истории каждого типа.

3. Перейдите в добавленный элемент.
4. Добавьте нужное количество баз данных (**База данных SePlatform.Historian**, **База данных Microsoft SQL Server** или **База данных PostgreSQL** в зависимости от типа сервера истории).
5. Для каждой базы данных в свойствах укажите:
   - **Тип использования** - какие данные хранятся в базе данных: значения и/или события
   - (только для SePlatform.Historian) **Псевдоним** - псевдоним базы данных, должен совпадать с псевдонимом, настроенным в SePlatform.Historian




## Сохранение данных

Для каждого SePlatform.Data Server, который должен сохранять данные:

1. Перейдите в его описание в домене (элемент **SePlatform.Server**).
2. Чтобы сохранять историю значений:
   2.1. Добавьте **Модуль истории** (есть по умолчанию).
   2.2. На изображении элемента нажмите **+** и в появившемся окне выберите базу данных.

   Можно выбрать несколько баз данных, находящихся в разных серверах истории: значения сигналов будут сохраняться во все добавленные базы данных.

   > ОБРАТИТЕ ВНИМАНИЕ
   > Не следует выбирать несколько баз данных, принадлежащих одному серверу истории. Это не решает задачу резервирования, поскольку при выходе из строя одного компонента будет потеряна связь со всеми его базами данных.

3. Чтобы сохранять историю событий:
   3.1. Добавьте модуль **OPC AE Сервер** (есть по умолчанию).
   3.2. На изображении элемента нажмите **+** и в появившемся окне выберите базу данных.

   Можно выбрать несколько баз данных, находящихся в разных серверах истории: события будут сохраняться во все добавленные базы данных.

   > ОБРАТИТЕ ВНИМАНИЕ
   > Не следует выбирать несколько баз данных, принадлежащих одному серверу истории. Это не решает задачу резервирования, поскольку при выходе из строя одного компонента будет потеряна связь со всеми его базами данных.

4. Объедините в сеть Ethernet компьютер, на котором находится **SePlatform.Server**, и компьютеры, на которых находятся серверы истории, в которые он сохраняет данные (стр. 51).

В результате SePlatform.Data Server будет сохранять значения сигналов и события в выбранные базы данных.

## Предоставление сохранённых данных

Чтобы SePlatform.Data Server предоставлял сохранённые им данные:

1. Перейдите в его описание в домене (элемент **SePlatform.Server**).
2. Добавьте **OPC HDA Сервер** (есть по умолчанию).

   В свойстве **Идентификатор сервера** укажите ProgId, по которому клиентские приложения смогут подключаться к SePlatform.Data Server по OPC HDA и запрашивать исторические данные.

   > ВАЖНО
   > ProgId должен быть уникальным как среди ProgId данного SePlatform.Data Server, так и среди ProgId других компонентов на том же компьютере.

3. Если значения сигналов и события сохраняются в разные базы данных, добавьте в **Модуль истории** те базы данных, в которые сохраняются события.

ПРИМЕЧАНИЕ

Это нужно сделать потому, что сохраняет события модуль **OPC AE Сервер**, а запрашивает - **Модуль истории**.

ОБРАТИТЕ ВНИМАНИЕ

Чтобы модуль истории использовал базу данных для запроса истории событий и не сохранял в неё значения сигналов, у базы данных в свойстве **Тип использования** выберите **«Для событий»**.

В результате SePlatform.Data Server будет предоставлять сохранённые им значения сигналов и события по OPC HDA. Если данные сохраняются в несколько баз данных, для запроса сохранённых данных будет использоваться та, с которой быстрее всего установлено соединение.

ОБРАТИТЕ ВНИМАНИЕ

Сохранённые события предоставляются по расширению OPC HDA: запрашивать их могут только клиентские приложения Систэм Платформ: SePlatform.Alarms, SePlatform.HMI и Service - OPCExplorer.

# 5.5. Агрегация событий

Агрегация событий - это объединение событий объекта в единую структуру. Использование агрегации позволяет:

- определять наличие активных и/или неквитированных событий в объекте.
- определять максимальную важность среди событий в объекте.
- квитировать сразу все активные события в объекте.

Для агрегации событий объекта, нужно вложить в него агрегатор - объект, настроенный для агрегации событий. Объект-агрегатор агрегирует события только того объекта, в который он вложен, не включая вложенные в него объекты. Чтобы агрегировать все события в ветке (в объекте и во всех вложенных объектах), нужно добавить агрегатор в каждый вложенный объект: они будут передавать агрегируемую информацию агрегатору, который находится в объекте выше.

ОБРАТИТЕ ВНИМАНИЕ

Чтобы информация передавалась между агрегаторами, они должны быть одного типа.

Чтобы агрегировать события объекта:

1. Создайте тип объекта-агрегатора:

1.1. Добавьте **Логический тип** и перейдите в него.

1.2. Добавьте сигналы. Тип элемента, описывающий сигнал, может быть любым. Тип значения каждого сигнала указывается в зависимости от того, какую информацию он агрегирует.

| **Агрегируемая информация** | **Тип значений** |
|---|---|
| **Информация о событиях** | |
| Наличие активных событий | bool |
| Наличие активных событий в ветке | |
| Все события квитированы | |
| Все события в ветке квитированы | |
| Максимальная важность актуальных событий | uint4 |
| Максимальная важность актуальных событий в ветке | |
| Максимальная важность активных событий | int2, uint2, int4, uint4, int8, uint8 |
| Максимальная важность активных событий в ветке | |
| Максимальная важность неквитированных событий | |
| Максимальная важность неквитированных событий в ветке | |
| Список идентификаторов активных объектов | string |
| **Команды** | |
| Квитирование событий | bool |
| Квитирование событий в ветке | |

1.3. У каждого добавленного сигнала в окне **События** установите флаг **Агрегировать события** и укажите, какую информацию сигнал агрегирует.

2. В объект, события которого нужно агрегировать, добавьте агрегатор:

2.1. Добавьте **Логический объект** созданного типа.

2.2. В окне **События** установите флаг **Агрегировать события**.

2.3. При необходимости настройте фильтрацию событий. Агрегироваться будут только те события, которые соответствуют настроенным фильтрам.

3. Чтобы агрегировать события в ветке, добавьте агрегатор того же типа в каждый вложенный объект (всех уровней вложенности).

# 5.6. Переопределение значений атрибутов с помощью карты атрибутов

ПРИМЕЧАНИЕ

Про атрибуты и работу с ними сказано в разделе 8.1.2. Атрибуты (стр. 167).

Значения атрибутов, добавленных типам и вложенным в них элементам, можно переопределить с помощью карты атрибутов. Карта атрибутов создаётся в приложении и позволяет изменить значения атрибута для элементов в этом приложении. Для каждого типа атрибута нужно создать свою карту значений атрибутов.

Чтобы создать карту атрибутов:

1. Перейдите в **Приложение**.
2. Добавьте элемент **Значения атрибутов**.
3. Создайте или выберите файл для хранения адресов:
   3.1. Нажмите на иконку папки на изображении элемента.
   Откроется окно выбора файла.
   3.2. Создайте или выберите файл.
   В результате:
   - Файл будет добавлен в список файлов проекта.
   - У элемента в свойствах будет указан выбранный файл.
4. В свойстве **Тип** укажите тип атрибута.
5. Перейдите в карту атрибутов.
   Откроется редактор значений атрибутов, в нём будут перечислены элементы в приложении, у которых есть атрибут указанного типа.
6. Элементам, для которых нужно переопределить значение атрибута, укажите новое значение атрибута.

# 5.7. Вычисление значений атрибутов

ПРИМЕЧАНИЕ

Про атрибуты и работу с ними сказано в разделе 8.1.2. Атрибуты (стр. 167).

В значении атрибута можно указать один или несколько параметров, которые будут вычислены при компиляции. Вычисляемый параметр указывается в атрибуте у типа или вложенного в тип элемента, когда нужно, чтобы в каждом объекте этого типа в атрибут подставилось своё значение вычисляемого параметра.

Вычисляемый параметр - это конструкция вида @(Source:Attribute), где Source - алгоритм поиска атрибута, Attribute - тип атрибута. При вычислении значение найденного атрибута будет подставлено вместо вычисляемого параметра.

Source может иметь следующие значения:

- не указан - атрибут ищется среди атрибутов данного элемента.
- **«object»** - атрибут ищется среди атрибутов контекстного объекта.

  Если вычисляемый параметр указан у сигнала, вложенного в тип, то у объектов этого типа атрибут будет искаться среди атрибутов объекта.
- **«*object»** - атрибут ищется среди атрибутов контекстного объекта и вверх по иерархии элементов в проекте: если у объекта не будет искомого атрибута, он будет искаться среди атрибутов родительского элемента и т.д..
- **«parent»** - атрибут ищется среди атрибутов родителя контекстного объекта (элемента, в который объект вложен).

  Если вычисляемый параметр указан у сигнала, вложенного в тип, то у объектов этого типа атрибут будет искаться среди атрибутов родителя объекта.

- **«*parent»** - атрибут будет искаться среди атрибутов родителя контекстного объекта и вверх по иерархии элементов в проекте: если у объекта не будет искомого атрибута, он будет искаться среди атрибутов его родительского элемента и т.д..

В Attribute нужно указать путь к типу атрибута в текущем проекте. Чтобы узнать путь к типу атрибута:

- в окне **Атрибуты** наведите мышью на атрибут нужного типа - путь отобразится во всплывающей подсказке.
- добавьте карту значений атрибутов и в свойстве **Тип** выберите атрибут нужного типа - в свойство будет подставлен путь к выбранному типу атрибута.

В Attribute можно использовать типы атрибутов из других модулей (например, **Om.System**) или пользовательские типы атрибутов. Пользовательские типы атрибутов создаются пользователем и предназначены для использования в вычислениях значений других атрибутов.

Чтобы создать пользовательский тип атрибута:

1. Перейдите в **Содержимое модуля** (корень проекта) или **Пространство имён**.
2. Добавьте **Тип атрибута**.
3. На изображении элемента или в свойстве **Тип значения** укажите тип значения.
4. На изображении элемента (поле **Заголовок**) или в свойстве **Название** укажите заголовок.

   Заголовок - это имя атрибута в окне **Атрибуты**.
5. (Опционально) На изображении элемента или в свойстве **Описание** укажите описание.

   Описание отображается в окне **Атрибуты** при наведении мышью на атрибут.
6. (Опционально) В свойстве **Значение по умолчанию** укажите значение, которое будет указываться при добавлении атрибута.

После создания пользовательского типа атрибута, элементам можно добавлять атрибуты этого типа.

ПРИМЕР

В тексте сообщения о событии нужно указать имя объекта, в котором произошло событие:

1. В типе сигналу настраиваем генерацию событий.
2. В окне **Атрибуты** в атрибуте **События** в текст сообщения вставляем имя объекта: **«@(object:System.Attributes.Name)»**.

В результате, в каждом объекте этого типа в текст сообщения подставится имя объекта.

ПРИМЕР

Для сигнала в типе настроена генерация событий. Нужно в некоторых объектах этого типа изменить важность события:

1. Создаём тип атрибута **«Severity»**.

   В свойствах указываем:

   - **Тип значения** - **«uint2»**
   - **Название** - **«Важность события»**

2. В типе сигналу настраиваем генерацию событий.
3. В окне **Атрибуты** в атрибуте **События** заменяем значение поля **Severity** на **«@(:Attributes.Severity)»** (нужно указать путь к типу атрибута, где он находится в проекте).
4. Добавляем сигналу атрибут созданного типа: **Важность события**. Указываем ему значение по умолчанию: **«500»**.
5. Для объектов этого типа, в которых нужно изменить важность события, переопределяем её с помощью карты атрибутов:

   5.1. Переходим в **Приложение**, в котором размещены объекты.

   5.2. Добавляем элемент **Значения атрибутов**, создаём файл карты атрибутов и указываем созданный тип атрибута - **Важность события**.

   5.3. В карте атрибутов объектам, в которых нужно, указываем новую важность события.

В результате:

- По умолчанию в объектах используется значение важности, указанное в типе.
- В объектах, где значение атрибута переопределено, используется значение важности, указанное в карте атрибутов.
- Все остальные параметры события (текст сообщения, звук и т.д.) имеют значения, указанные в типе.

ПРИМЕР

Для сигнала в типе настроена генерация событий. В каждом объекте этого типа должен быть свой текст сообщения о событии:

1. Создаём тип атрибута **«Message»**.

   В свойствах указываем:

   - **Тип значения** - **«string»**
   - **Название** - **«Сообщение о событии»**

2. В типе сигналу настраиваем генерацию событий.
3. В окне **Атрибуты** в атрибуте **События** заменяем текст сообщения на **«@(object:Attributes.Message)»** (нужно указать путь к типу атрибута, где он находится в проекте).
4. Объектам этого типа добавляем атрибут созданного типа: **Сообщение о событии**. Значение атрибута - текст сообщения.

В результате в каждом объекте событие будет генерироваться с текстом, указанным в атрибуте **Сообщение о событии** этого объекта.

# 5.8. SePlatform.AccessPoint: создание привязок

Привязка - это ссылка в SePlatform.AccessPoint, которая указывает на объект или приложение, размещённое в SePlatform.Data Server. В процессе работы SePlatform.AccessPoint привязка динамически раскрывается: в дереве сигналов SePlatform.AccessPoint создаётся папка, в которой

создаётся копия дерева сигналов объекта или приложения, на которое указывает привязка. При изменении значений сигналов в SePlatform.Data Server, новые значения автоматически передаются и записываются в соответствующие сигналы в SePlatform.AccessPoint.

ПРИМЕЧАНИЕ

Привязки подчёркивают роль SePlatform.AccessPoint в качестве точки доступа к данным. В то время как экземпляры SePlatform.Data Server используются для сбора и обработки данных, SePlatform.AccessPoint работает на АРМ оператора, с помощью привязок получает данные одного или нескольких экземпляров SePlatform.Data Server и предоставляет их клиентским приложениям.

Чтобы создать привязку:

1. Перейдите в **Приложение** или в **Тип приложения**, в котором описываются данные для SePlatform.AccessPoint: см. раздел 5.1.3. Добавление приложений (стр. 44).
2. Добавьте ссылку на объект или приложение, размещённое в SePlatform.Data Server.

   Добавление ссылок описано в разделе Добавление вложенных объектов и ссылок (стр. 64).

   Укажите имя ссылки - оно станет именем папки, в которой будет создаваться дерево сигналов.
3. Ссылке добавьте атрибут **Раскрывать ссылку динамически**.

   Атрибут находится в модуле **Om.Server**. Добавление атрибутов описано в разделе 8.1.2. Атрибуты (стр. 167).
4. Настройте передачу данных между SePlatform.Data Server и SePlatform.AccessPoint по протоколу TCP: см. раздел 5.1.4. Настройка передачи данных (стр. 48).

ОБРАТИТЕ ВНИМАНИЕ

Привязки не работают через файловый интерфейс. Для передачи данных используйте сеть Ethernet.

# 5.9. Чтение и запись значений по одному адресу

Если параметр находится вне SePlatform.Data Server, для SePlatform.Data Server можно настроить передачу значений этого параметра в двух направлениях: на чтение и на запись.

ПРИМЕР

В ПЛК задана некоторая уставка. В SePlatform.Data Server нужно как получать текущее значение этой уставки, так и изменять его по команде пользователя.

В этом случае:

- Исходный параметр будет один.

  Адрес[1] этого параметра будет использоваться как для чтения, так и для записи значений.
- В SePlatform.Data Server исходному параметру будут соответствовать два параметра: один для получения текущего значения и один для передачи команд на изменение значения.

[1] В исполняемом компоненте, в котором будет размещён параметр.

ОБРАТИТЕ ВНИМАНИЕ

Чтение и запись значений по одному адресу возможны по протоколам/спецификациям:

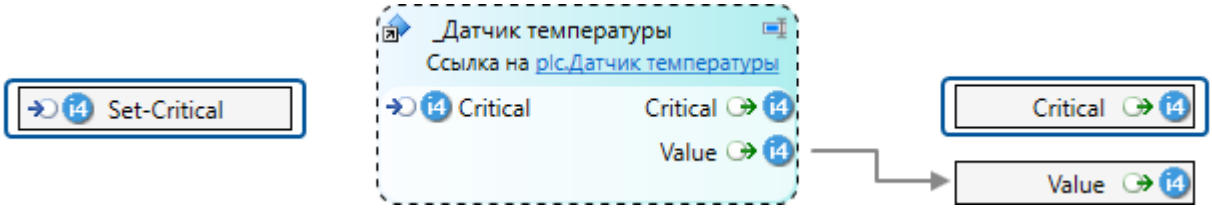
- OPC DA
- OPC UA
- Modbus
- FINS
- UNET

Для параметра, значения которого нужно читать и писать:

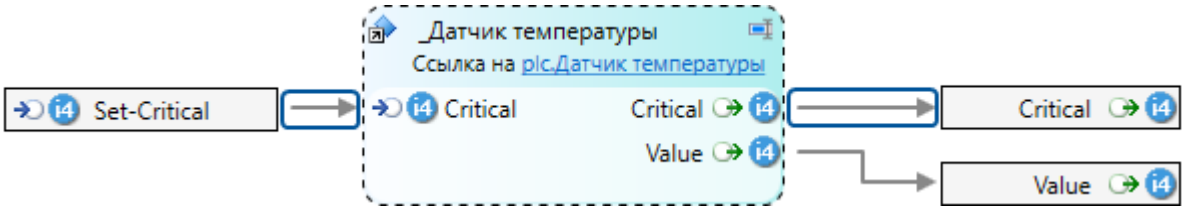
1. В SePlatform.Data Server добавьте два параметра.

ПРИМЕЧАНИЕ

Для получения текущего значения рекомендуем использовать элемент **Параметр**, а для передачи команд - элемент **Событие**.



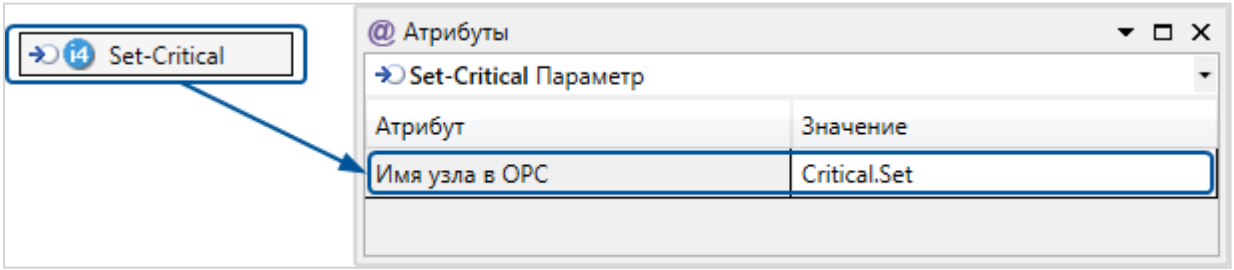

2. Свяжите добавленные параметры с параметром-источником.



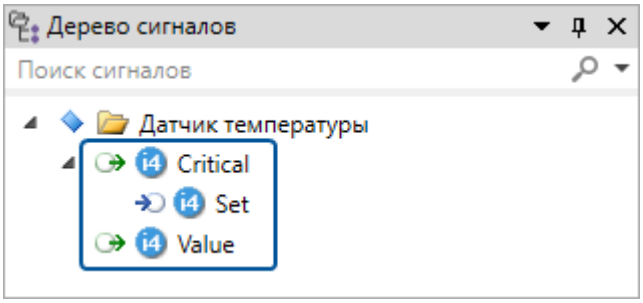

3. (Опционально) Чтобы в дереве сигналов добавленные параметры были представлены в иерархическом виде:

3.1. Параметру, который используется для записи, добавьте атрибут **`Имя узла в OPC`**.

3.2. В атрибуте укажите значение <имя параметра>.Set.




В результате дерево сигналов будет иметь вид:



# 5.10. Передача метки времени по Modbus

В протоколе Modbus нет стандартного способа передачи метки времени. По умолчанию значениям, полученным по Modbus, SePlatform.Data Server в качестве метки времени присваивает момент получения значения.

Если требуется передавать метку времени, можно настроить её передачу через регистры: устройство будет записывать в регистры значение вместе с его меткой времени, а SePlatform.Data Server будет получать значение, его метку времени и присваивать полученную метку времени этому значению.

Для использования данной возможности необходимо в устройстве, передающем данные по Modbus, реализовать запись метки времени в регистры памяти в формате, описанном ниже (стр. 106).

ОБРАТИТЕ ВНИМАНИЕ

Передача метки времени поддерживается для типов:

- int2, uint2
- int4, uint4
- float
- double

## Формат метки времени

Метка времени занимает три регистра памяти, следующих за регистром(ами) значения.

| Рег./Поз. | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1-4 | Данные занимают от 1 до 4 регистров в зависимости от типа сигнала | | | | | | | | | | | | | | | |
| 5 | секунды | | | | | | | | миллисекунды | | | | | | | |
| 6 | час | | | | | | | | минуты | | | | | | | |
| 7 | год | | | | | | | | месяц | | | | день | | | |

## Как настроить передачу метки времени

Чтобы настроить передачу метки времени, нужно в карте адресов Modbus:

- Для параметра указать, что он передаётся с меткой времени.
- Не использовать три регистра, следующих за регистром(ами) значения параметра, для приёма/передачи других значений. Эти регистры будут использоваться для передачи метки времени.

Задавать адреса можно:

- Вручную (или сторонней программой).
- Или с помощью автозаполнения карты адресов.

### Ручное заполнение карты адресов

Карта адресов заполняется вручную (или с помощью сторонней программы), если адреса уже известны (определены в программе контроллера или описаны в некотором документе), и их нужно указать в проекте

SePlatform.Development Studio.

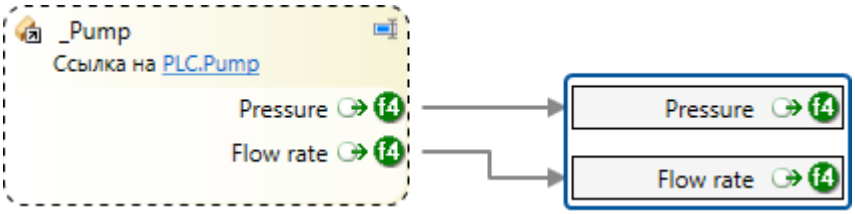
При настройке адресов для каждого параметра, значения которого будут передаваться с меткой времени, в поле **Метка времени** укажите **«Да»**.

| | Сигнал | Тип | Привязка | Сегмент | Адрес | Номер бита | ıер записи в фа | Метка времени | Размер строки | этегория данны |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | |
| f4 | Pump.Pressure | float | непосредственно | Holding Registers | 0 | | | Да | | |
| f4 | Pump.Flow rate | float | непосредственно | Holding Registers | 5 | | | Да | | |

В результате при получении значения SePlatform.Data Server будет получать также его метку времени (следующие три регистра после регистра(ов) значения) и присваивать её полученному значению.

ПРИМЕР

При описанной выше настройке, в SePlatform.Data Server в параметры *Pressure* и *Flow rate* значения будут записываться с меткой времени, полученной от устройства.




## Автоматическое заполнение карты адресов

Автоматическое заполнение карты адресов используется, если описание проекта создаётся в SePlatform.Development Studio, а программа для устройства создаётся на основе этого описания. В этом случае адреса параметров для устройства будут взяты из проекта SePlatform.Development Studio, а чтобы в проекте SePlatform.Development Studio не задавать адреса вручную, можно использовать автоматическое заполнение карты адресов.

Поскольку адреса параметров в устройстве будут получены из проекта SePlatform.Development Studio, то помимо параметра-значения нужно описать также параметры метки времени: их адреса будут использоваться в устройстве для записи метки времени.

ПРИМЕЧАНИЕ

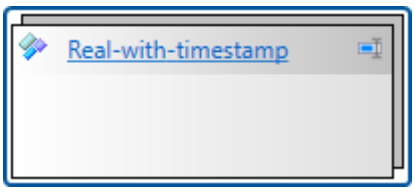
Если программа для устройства разрабатывается без использования SePlatform.Development Studio, то параметры метки времени добавлять не нужно, поскольку в проекте SePlatform.Development Studio они не используются. Поэтому при ручном заполнении карты адресов они не добавляются.

Параметры значения и его метки времени рекомендуем объединять в сокет: этот сокет будет описывать исходный параметр, а вложенные в него параметры будут значением и меткой времени исходного параметра.

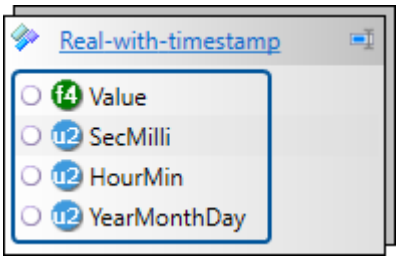
Поскольку таких исходных параметров (которые передаются с меткой времени) может быть сколь угодно много, для описания рекомендуем использовать тип сокета.

Рекомендуемый способ настройки:

1. Опишите сокет:
    1.1. Добавьте в проект **Тип сокета**.




1.2. В него добавьте параметры.




| Параметр | Тип | Описание |
|---|---|---|
| Value | int2, uint2<br>int4, uint4<br>float<br>double | Передаваемое значение. |
| SecMilli | int2, uint2 | Часть метки времени, содержащая секунды и миллисекунды. |
| HourMin | int2, uint2 | Часть метки времени, содержащая час и минуты. |
| YearMonthDay | int2, uint2 | Часть метки времени, содержащая день, месяц и год. |

ОБРАТИТЕ ВНИМАНИЕ

Добавлять параметры нужно в том порядке, в котором они перечислены: это нужно для того, чтобы при автозаполнении карты адресов им были назначены корректные адреса.

ПРИМЕЧАНИЕ

Имена параметрам можно указать любые.

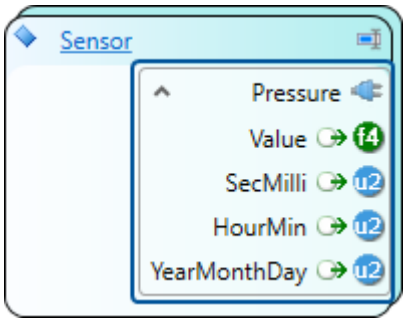
ПРИМЕЧАНИЕ

Параметрам метки времени можно указать любой тип из перечисленных.

2. На стороне ПЛК:

2.1. Перейдите в объект (или тип объекта), в который нужно добавить параметр.

2.2. Добавьте **Сокет** описанного типа.



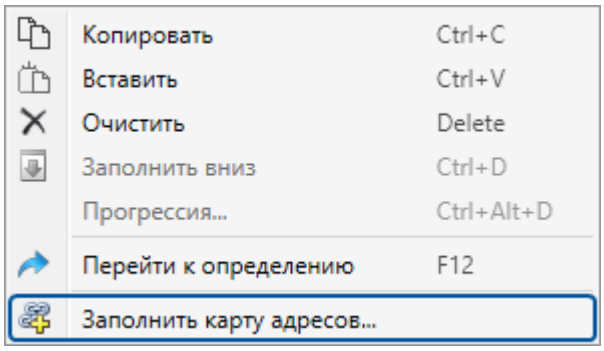

ПРИМЕЧАНИЕ

Советуем задать сокету имя того параметра, который он описывает.

2.3. В карте адресов:

2.3.1. Укажите адреса параметров.

Для этого в контекстном меню выберите **Заполнить карту адресов...**.




Откроется мастер заполнения карты адресов. После того, как настроите параметры, нажмите кнопку **Готово**.

В результате каждому параметру будет присвоен адрес. Параметрам метки времени будут назначены три регистра следующие за параметром-значением.

| Сигнал | Тип | Привязка | Сегмент | Адрес | Номер бита | ...ер записи в фа... | Метка времени | Размер строки | ...тегория данны... |
|---|---|---|---|---|---|---|---|---|---|
| Sensor.Pressure.Value | float | непосредственно | Input Registers | 0 | | | Нет | | |
| Sensor.Pressure.SecMilli | uint2 | непосредственно | Input Registers | 2 | | | Нет | | |
| Sensor.Pressure.HourMin | uint2 | непосредственно | Input Registers | 3 | | | Нет | | |
| Sensor.Pressure.YearMonthDay | uint2 | непосредственно | Input Registers | 4 | | | Нет | | |

ОБРАТИТЕ ВНИМАНИЕ

Если вы добавляли параметр-значение и параметры метки времени не в том порядке, как описано выше, то адреса им будут присвоены некорректно. В этом случае потребуется проверить и при необходимости исправить присвоенные адреса.

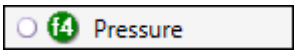
2.3.2. Параметру-значению в поле **Метка времени** установите **«Да»**.

| Сигнал | Тип | Привязка | Сегмент | Адрес | Номер бита | ...ер записи в фа... | Метка времени | Размер строки | ...тегория данны... |
|---|---|---|---|---|---|---|---|---|---|
| Sensor.Pressure.Value | float | непосредственно | Holding Registers | 0 | | | Да | | |
| Sensor.Pressure.SecMilli | uint2 | непосредственно | Holding Registers | 2 | | | Нет | | |
| Sensor.Pressure.HourMin | uint2 | непосредственно | Holding Registers | 3 | | | Нет | | |
| Sensor.Pressure.YearMonthDay | uint2 | непосредственно | Holding Registers | 4 | | | Нет | | |

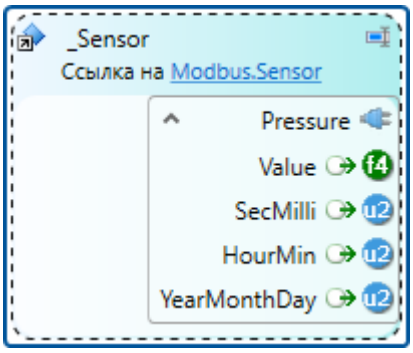
3. На стороне SePlatform.Data Server:

3.1. Перейдите в объект (или тип объекта), в который будут передаваться значения параметра.

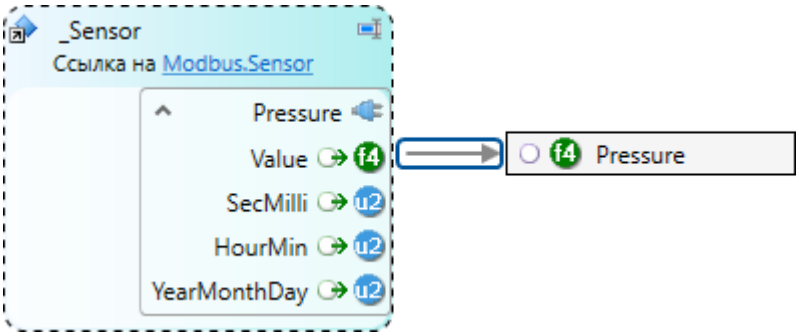
3.2. Добавьте параметр, в который будут записываться полученные значения.




3.3. Добавьте ссылку на объект в устройстве (в ПЛК), если такой ссылки ещё нет.




3.4. Добавьте связь от параметра-значения в сокете на изображении ссылки к параметру-приёмнику.




ПРИМЕЧАНИЕ

Настраивать передачу параметров метки времени не нужно. Получать и преобразовывать их значения в метку времени SePlatform.Data Server будет автоматически.

В результате:

› В проекте описаны адреса самого значения и адреса его метки времени: эти адреса можно использовать в программе устройства.

› SePlatform.Data Server при получении значений параметра, будет получать также метку времени этого значения и записывать в сигнал значение с присвоенной ему меткой времени.

# 5.11. Передача данных по категориям

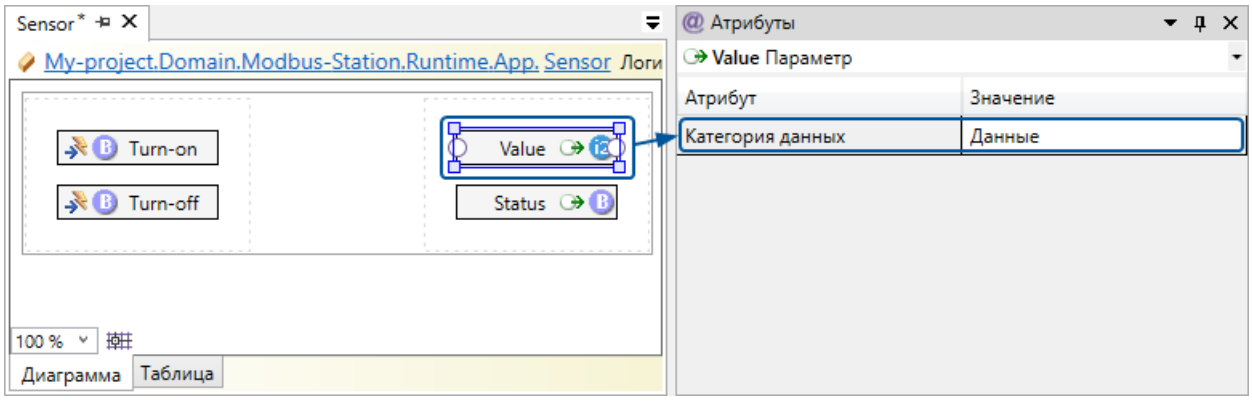
Данные, передаваемые между компонентами, можно разделить на категории и передавать данные разных категорий с различной скоростью. Это используется, когда объем передаваемых данных большой и они не могут быть переданы за приемлемое время, при этом есть важные данные, которые нужно передать быстро, а также данные, задержка при передаче которых не скажется на работоспособности системы.
У логических адаптеров **Опросчик Modbus TCP** и **Опросчик Modbus RTU** период опроса категорий данных можно задать в настройках адаптера. Для остальных логических адаптеров нужно разделить потоки данных и передавать категории данных в разных потоках (стр. 112).

# 5.11.1. Категории данных Modbus

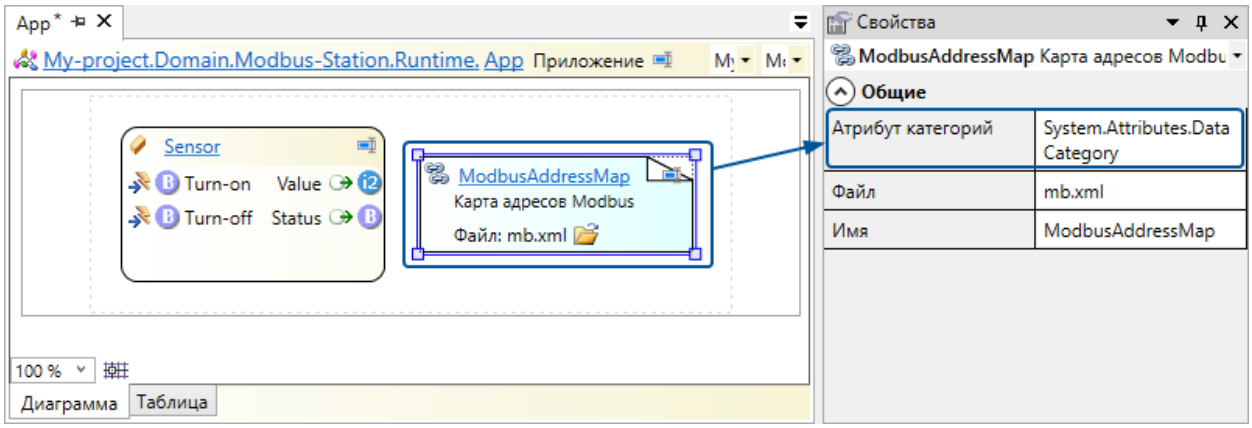
1. У подчинённой станции, которую опрашивает логический адаптер:

   1.1. Сигналам добавьте атрибут **Категория данных** или любой другой атрибут со строковым значением.

   В качестве значения атрибута каждому сигналу укажите произвольное имя категории.




   1.2. У элемента **Карта адресов Modbus** в свойстве **Атрибут категорий** укажите атрибут.




   При переходе в карту адресов в столбце **Категория данных** для каждого сигнала будет показано значение атрибута. Значение можно переопределить в карте адресов.

| | Сигнал | Тип | Привязка | Сегмент | Адрес | Номер бита | Категория данных |
|---|---|---|---|---|---|---|---|
| B | Sensor.Turn-on | bool | непосредственно | Coils | 0 | 0 | Команды |
| B | Sensor.Turn-off | bool | непосредственно | Coils | 1 | 0 | Команды |
| i2 | Sensor.Value | int2 | непосредственно | Holding Registers | 0 | | Данные |
| B | Sensor.Status | bool | непосредственно | Discretes Input | 0 | 0 | Статистика |

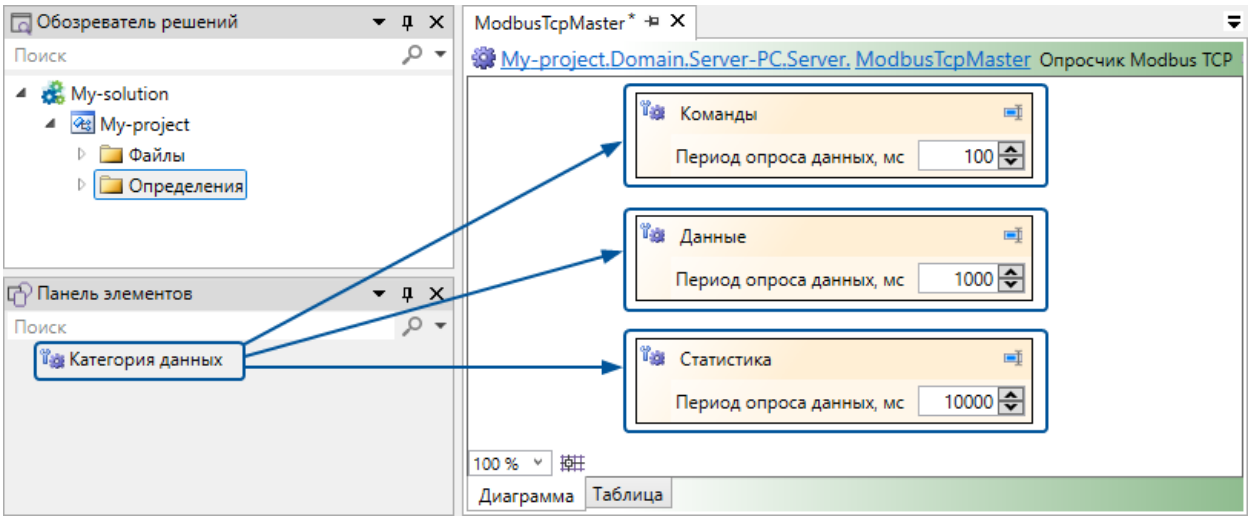
2. У логического адаптера, который опрашивает станцию (**Опросчик Modbus TCP** или **Опросчик Modbus RTU**):

2.1. Перейдите в логический адаптер.

2.2. Для каждой категории данных добавьте элемент **Категория данных**.

Элементу укажите:

- **Имя** - название категории данных.
- **Период опроса данных, мс** - период опроса.




В результате логический адаптер будет запрашивать разные категории данных с разным периодом.

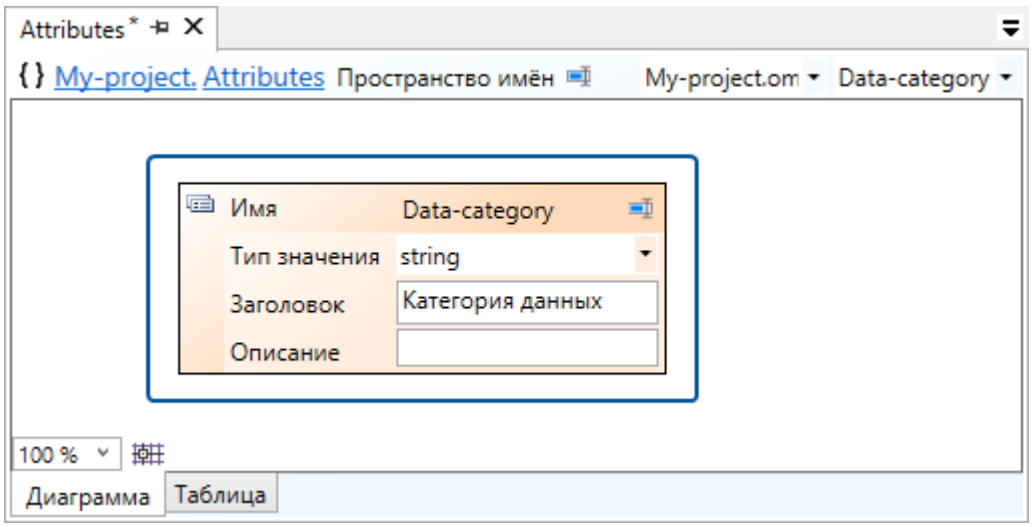
## 5.11.2. Разделение потоков

Чтобы разделить потоки, нужно:

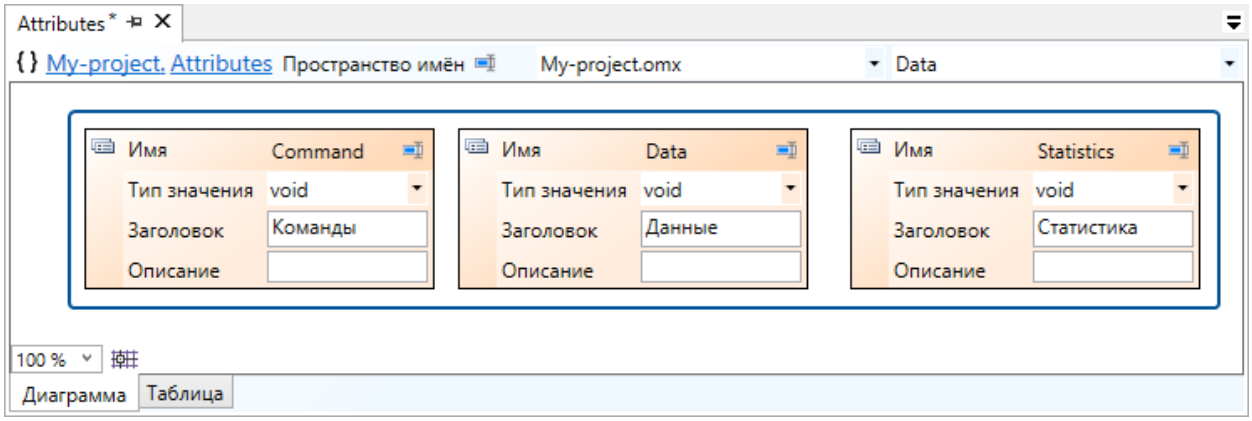
1. Указать категорию данных для сигналов, значения, которых передаются (стр. 112).
2. Разделить потоки данных. Разделить потоки можно:
   - На стороне клиента (стр. 114) - в компоненте, который запрашивает или получает данные.
   - На стороне сервера (стр. 118) - в компоненте, который предоставляет или передаёт данные.

### Указание категории данных

1. Создайте тип(ы) атрибута(ов) для указания категорий данных:
   - Вариант 1. Создайте один строковый тип атрибута: категория данных будет указываться в значении этого атрибута.




› Вариант 2. Для каждой категории данных создайте пустой тип атрибута: категория данных будет определяться наличием атрибута созданного типа.




Чтобы создать тип атрибута:

1.1. Перейдите в **Содержимое модуля** (корень проекта) или **Пространство имён**.

1.2. Добавьте **Тип атрибута**.

1.3. На изображении элемента или в свойстве **Тип значения** укажите тип значения - **string** (строковый) или **void** (пустой).

1.4. На изображении элемента (поле **Заголовок**) или в свойстве **Название** укажите заголовок - имя атрибута в окне **Атрибуты**. Для строкового - произвольное имя (например, **«Категория данных»**), для пустого типа атрибута - название категории (например, **«Команды»**).

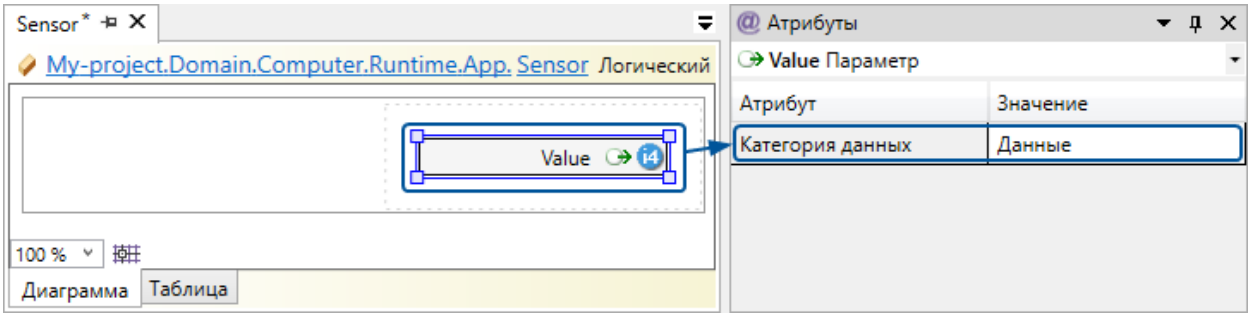
1.5. (Опционально) На изображении элемента или в свойстве **Описание** укажите описание.

Описание отображается в окне **Атрибуты** при наведении мышью на атрибут.

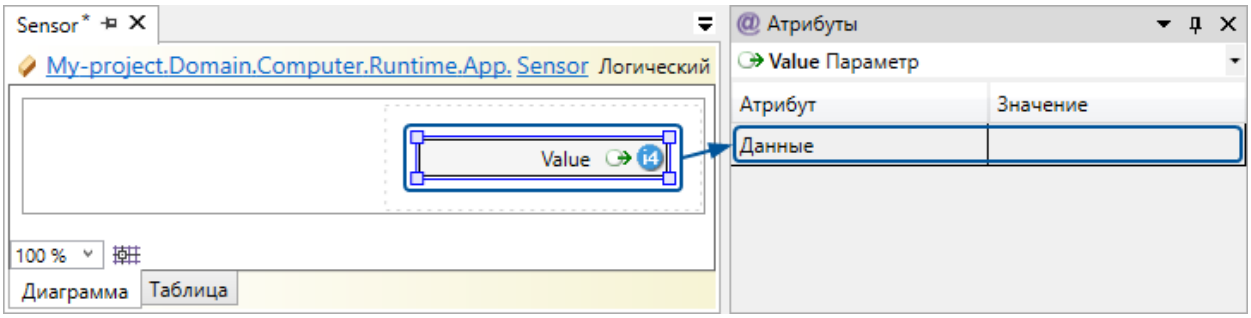
1.6. (Опционально) Для строкового типа атрибута в свойстве **Значение по умолчанию** укажите значение, которое будет указываться при добавлении атрибута.

2. Сигналам, значения которых передаются, укажите категорию:

› Если используется строковый атрибут - сигналу добавьте атрибут и в значении атрибута укажите название категории.



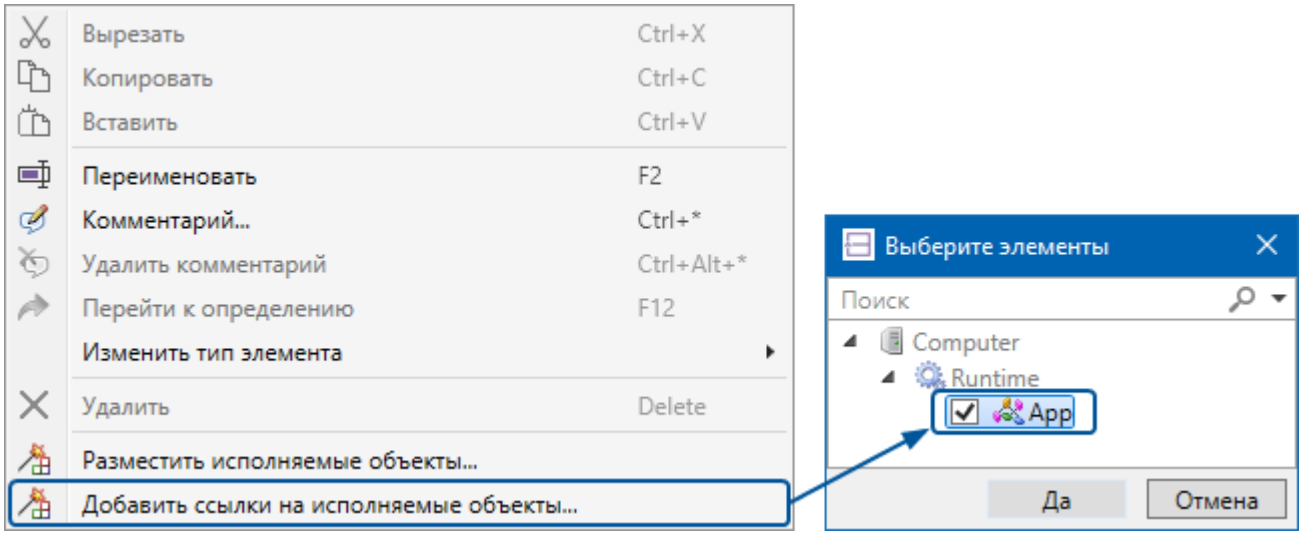

› Если используются пустые атрибуты - сигналу добавьте атрибут нужного типа.




## Разделение потоков на стороне клиента

1. Перейдите в компонент-приёмник (компонент, который запрашивает или получает данные).

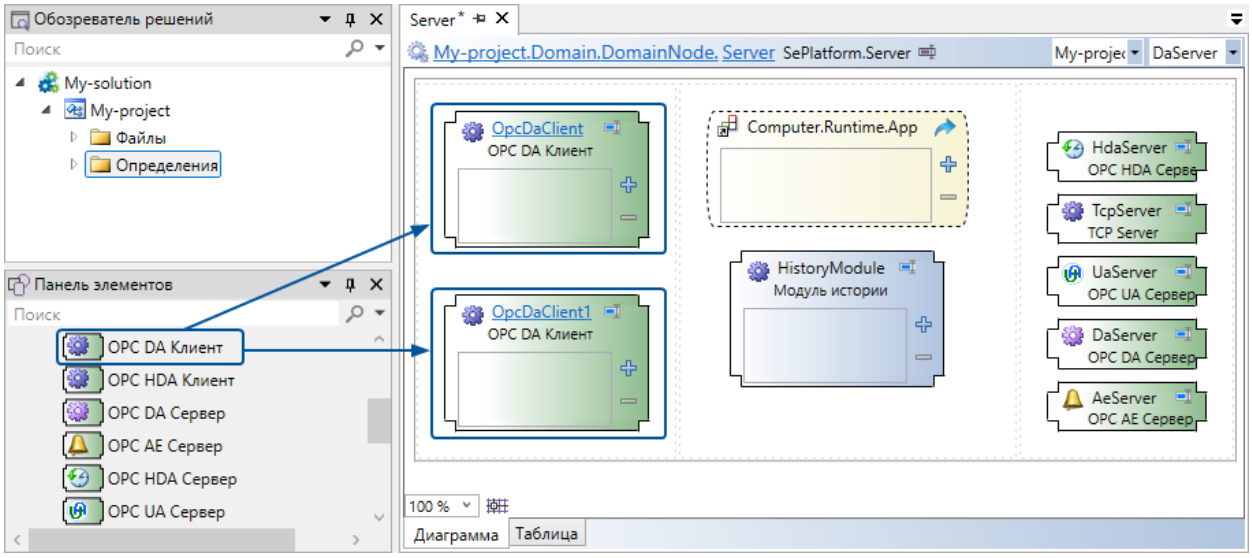
2. В контекстном меню выберите **Добавить ссылки на исполняемые объекты...** и в появившемся окне выберите приложение, в котором находятся передаваемые данные.




Будет добавлена **Ссылка на исполняемый модуль приложения**, указывающая на выбранное приложение.

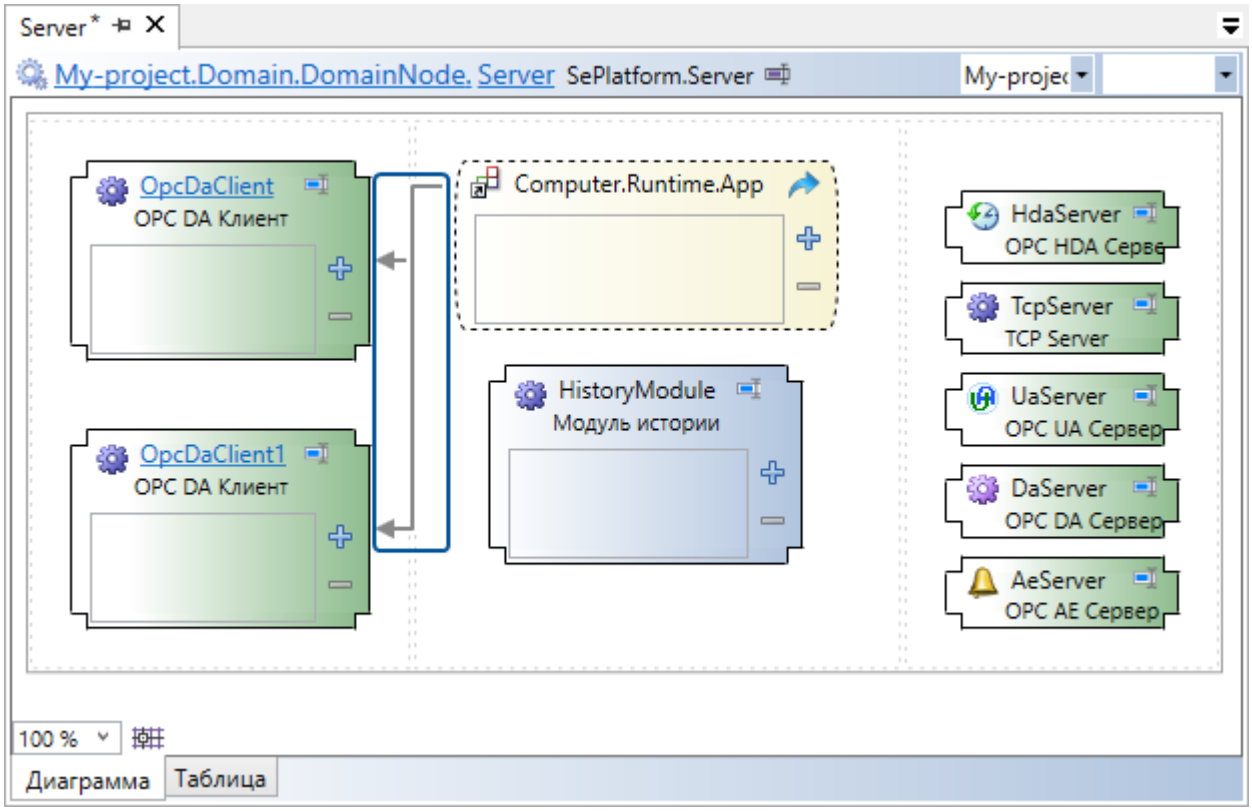
3. Для каждой категории данных:

3.1. Добавьте клиентский логический адаптер для получения данных от компонента-источника.



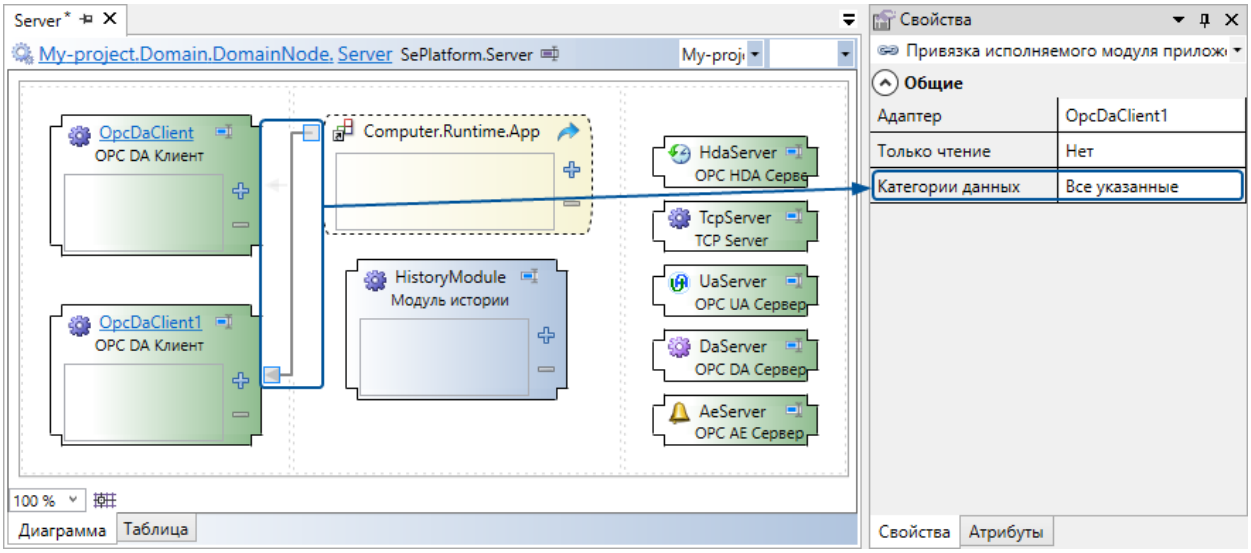

Адаптеры, используемые для получения данных по каждому протоколу, перечислены в разделе 5.1.4. Настройка передачи данных (стр. 48).

3.2. Протяните связь от элемента **Ссылка на исполняемый модуль приложения** к добавленному адаптеру.



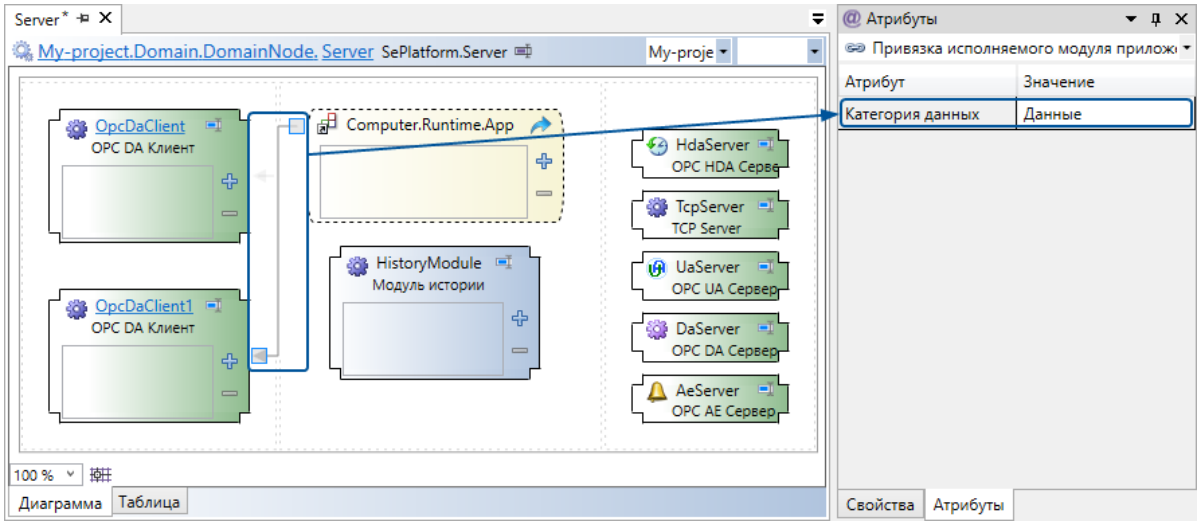
3.3. У связи в свойстве **Категории данных** укажите:

- **«Все»** - по связи будут передаваться все данные
- **«Все указанные»** - по связи будут передаваться только указанные категории данных.
- **«Все, кроме указанных»** - по связи будут передаваться все данные, кроме указанных категорий, в том числе данные без категории.

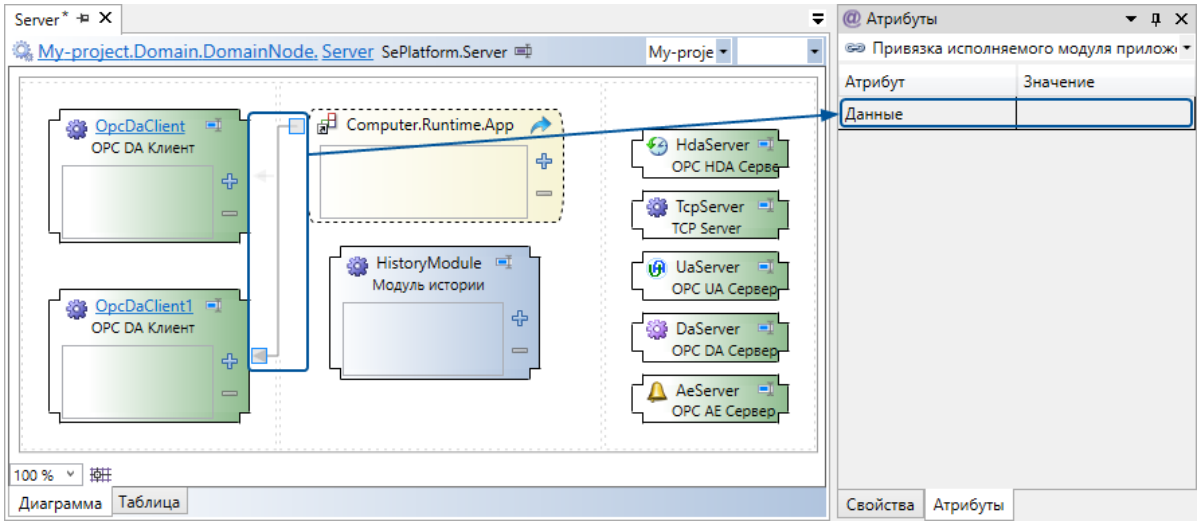


3.4. Для связи укажите категорию данных:

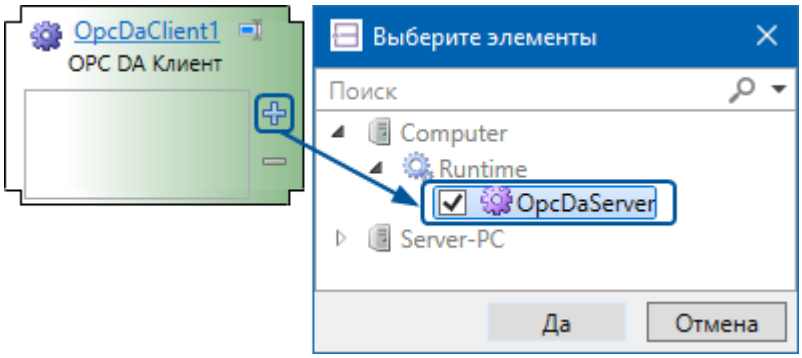
› Если используется строковый атрибут - связи добавьте атрибут и в значении атрибута укажите название категории.




› Если используются пустые атрибуты - связи добавьте атрибут нужного типа. Можно добавить несколько атрибутов разных типов.

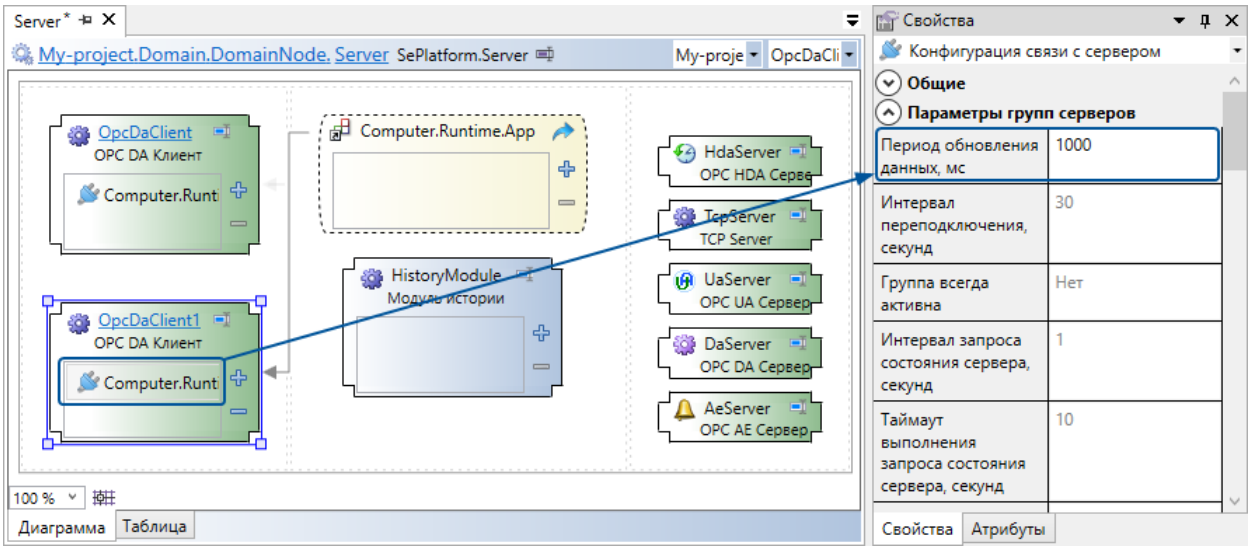



3.5. На изображении адаптера нажмите + и в появившемся окне выберите серверный логический адаптер в составе компонента-источника.




В адаптер будет добавлен элемент **Конфигурация связи с сервером**.

3.6. В свойствах элемента укажите период обновления данных.




В результате:

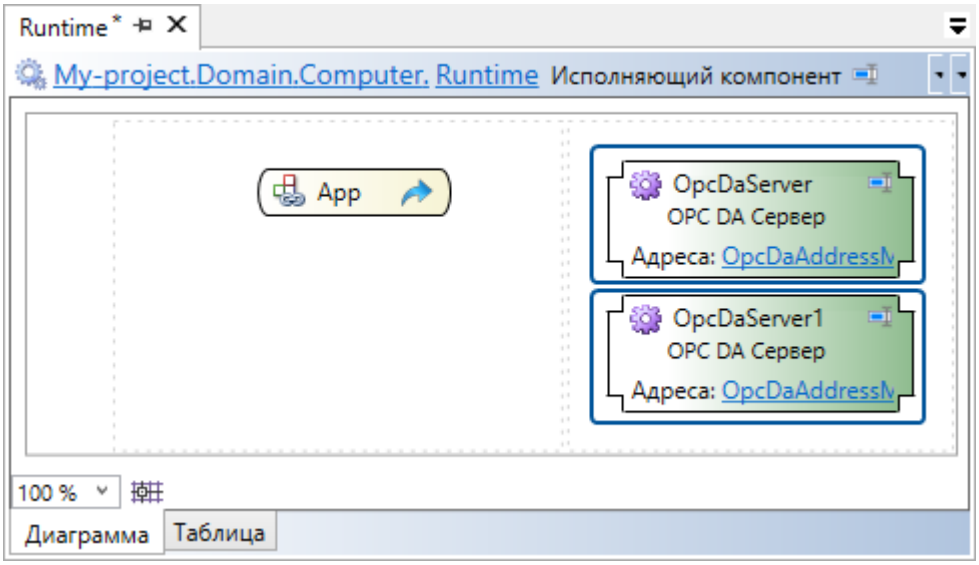
- В компоненте-приёмнике будет несколько клиентских логических адаптеров.
- Разные логические адаптеры будут запрашивать/получать от компонента-источника разные категории данных и с разным периодом.

## Разделение потоков на стороне сервера

ОБРАТИТЕ ВНИМАНИЕ

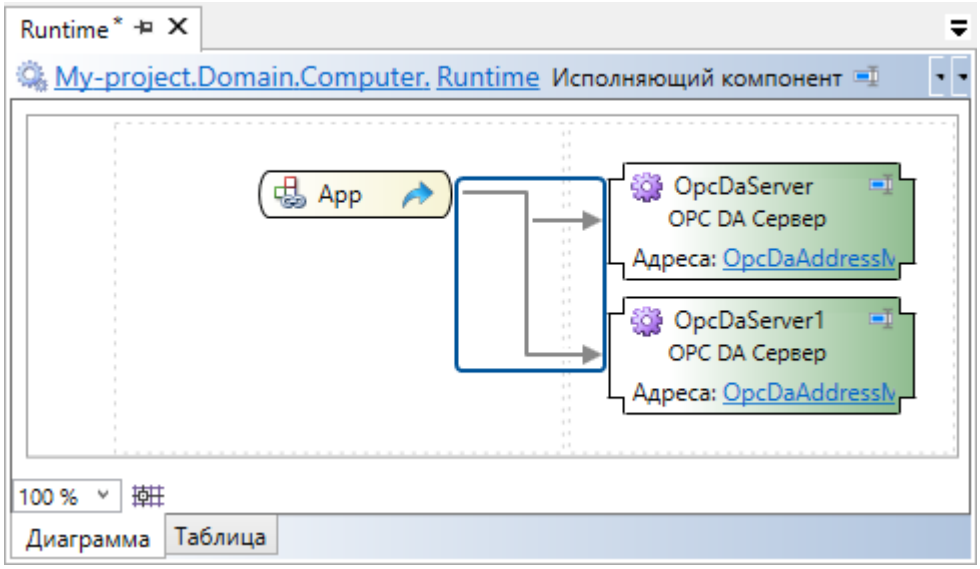
Для разделения потоков на стороне сервера нужно, чтобы в компоненте-источнике было расположено не **Приложение**, а **Исполняемый модуль приложения** (см. раздел 5.1.3. Добавление приложений (стр. 44)), поскольку для приложения нельзя настроить разделение потоков.

1. Перейдите в компонент-источник (компонент, который предоставляет или передаёт данные).
2. Для каждой категории данных:
   2.1. Добавьте серверный логический адаптер.



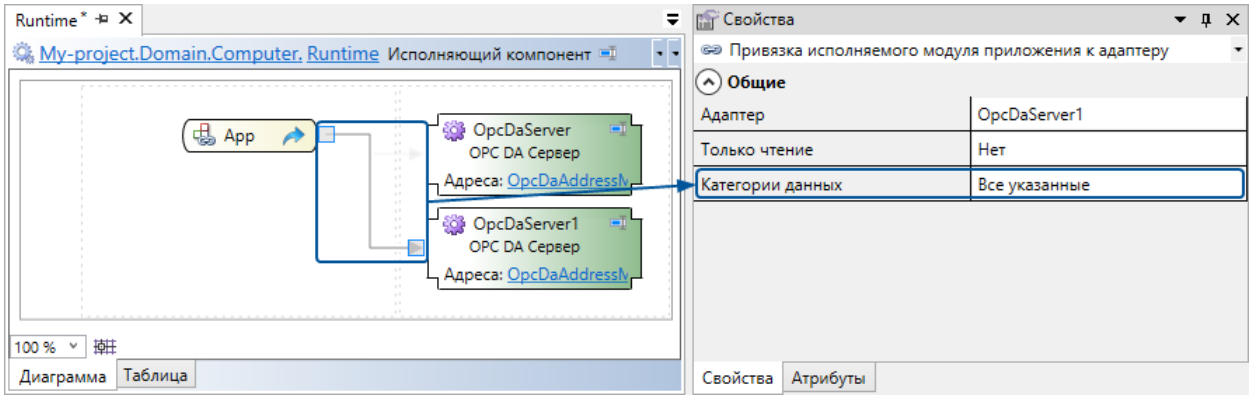

Адаптеры, используемые для передачи данных по каждому протоколу, перечислены в разделе 5.1.4. Настройка передачи данных (стр. 48).

2.2. Протяните связь от элемента **Исполняемый модуль приложения** к добавленному адаптеру.



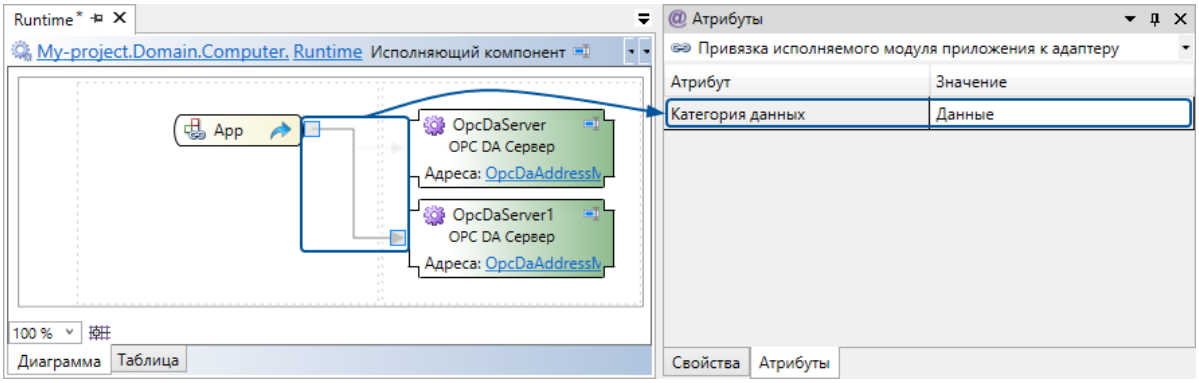

2.3. У связи в свойстве **Категории данных** укажите:

- **«Все»** - по связи будут передаваться все данные
- **«Все указанные»** - по связи будут передаваться только указанные категории данных.
- **«Все, кроме указанных»** - по связи будут передаваться все данные, кроме указанных категорий, в том числе данные без категории.

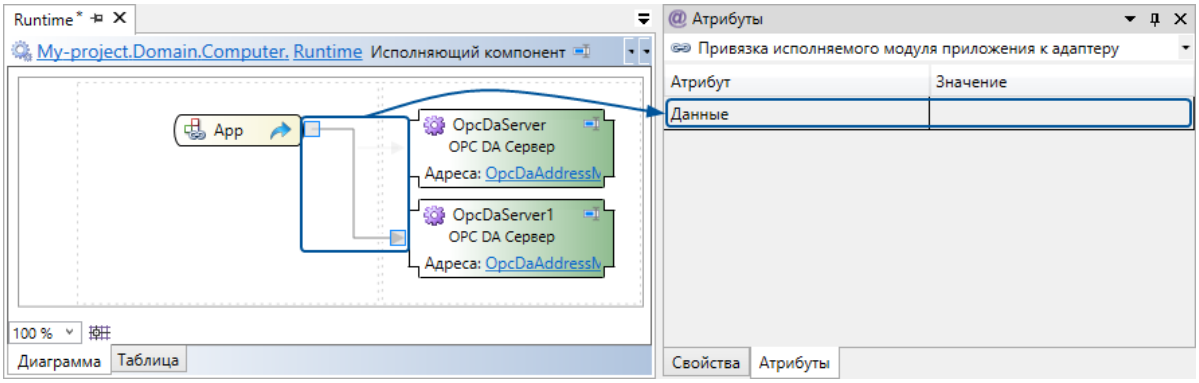


2.4. Для связи укажите категорию данных:

› Если используется строковый атрибут - связи добавьте атрибут и в значении атрибута укажите название категории.

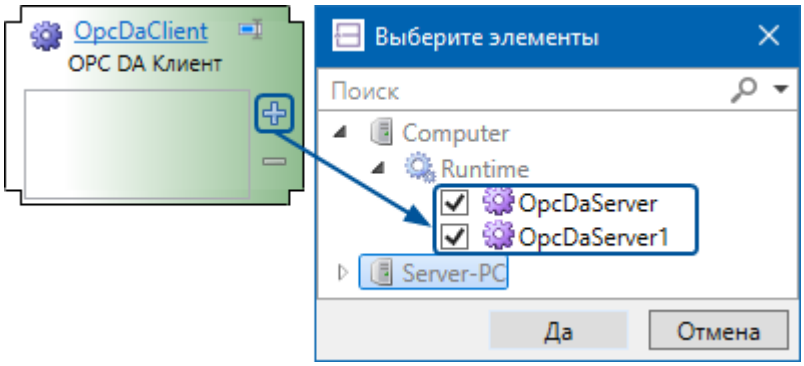



› Если используются пустые атрибуты - связи добавьте атрибут нужного типа. Можно добавить несколько атрибутов разных типов.



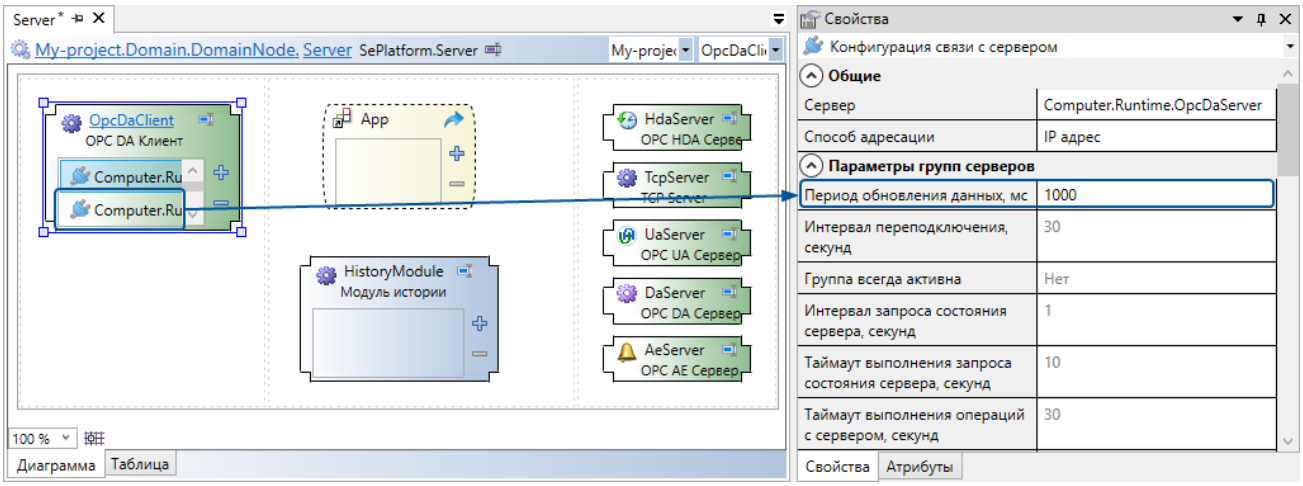

3. Перейдите в компонент, который запрашивает или получает данные.

4. На изображении клиентского логического адаптера нажмите + и в появившемся окне выберите серверные логические адаптеры в составе компонента-источника.




В адаптер будут добавлены конфигурации связи с выбранными адаптерами.

5. В свойствах каждой конфигурации связи укажите период обновления данных.




В результате:

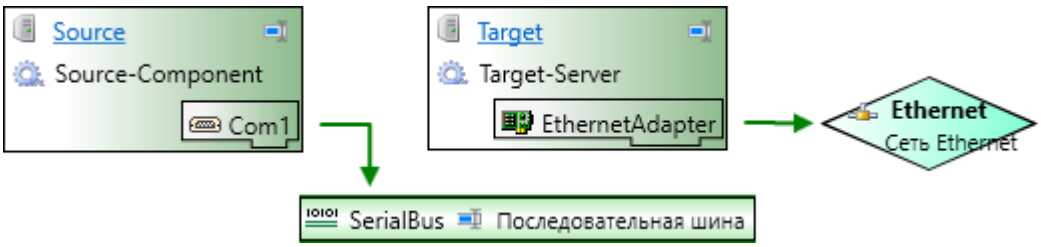
- Компонент-источник будет предоставлять/передавать разные категории данных через разные серверные логические адаптеры.
- Компонент-приёмник будет запрашивать/принимать данные через один клиентский логический адаптер, но с разным периодом опроса разных серверных логических адаптеров.

# 5.12. Передача данных между компонентами, находящимися в разных сетях

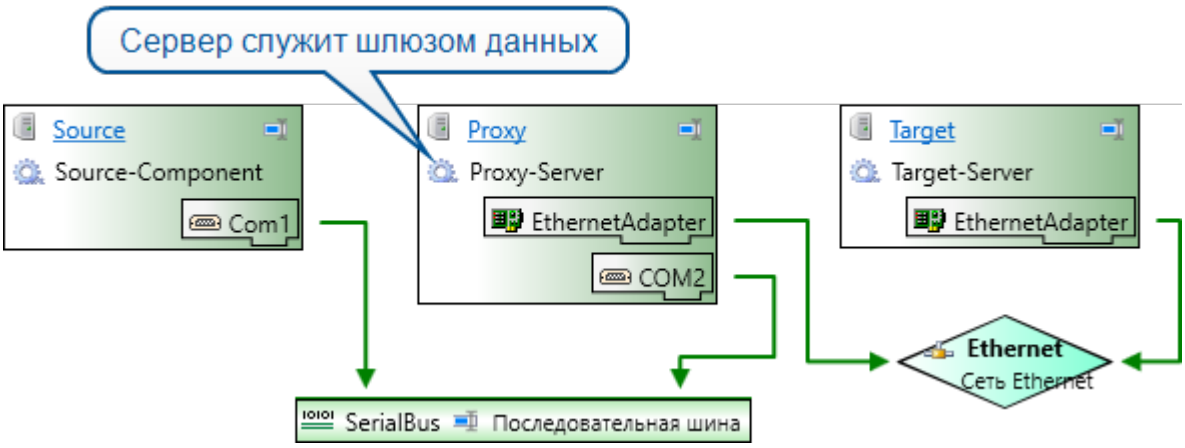
Два компонента, которые должны передавать друг другу данные, могут находиться в разных сетях. Такая ситуация может возникнуть, если:

- компоненты находятся в разных логических сетях, между которыми работает брандмауэр.


- компоненты находятся в разных физических сетях.




В этом случае компоненты не могут передавать друг другу данные напрямую. Чтобы организовать передачу данных между ними, нужно на компьютер, подключённый к обеим сетям, добавить SePlatform.Data Server и использовать его в качестве промежуточного узла - шлюза данных. Данные будут передаваться по схеме: Источник данных → Шлюз данных → Получатель данных.




Шлюз данных можно использовать при передаче данных по протоколам:

- TCP - в сети Ethernet или с помощью файлового обмена.
- Modbus - в сети Ethernet (Modbus TCP) или по последовательной шине (Modbus RTU).

ОБРАТИТЕ ВНИМАНИЕ

Источник и получатель данных должны соединяться со шлюзом данных по одному и тому же протоколу.

Для передачи данных используются следующие логические адаптеры:

- Источник данных → шлюз данных:

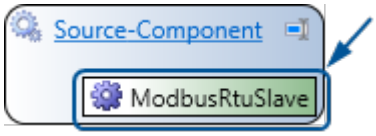
| Протокол | Сеть | Адаптер источника | Адаптер шлюза |
|---|---|---|---|
| TCP | Сеть Ethernet<br>Файловый обмен | TCP Server | HUB Module |
| Modbus | Сеть Ethernet | Станция Modbus TCP | Опросчик Modbus TCP |
| | Последовательная шина | Станция Modbus RTU | Опросчик Modbus RTU |

- Шлюз данных → Получатель данных

| Протокол | Сеть | Адаптер получателя | Адаптер шлюза |
|---|---|---|---|
| TCP | Сеть Ethernet<br>Файловый обмен | HUB Module | TCP Server |
| Modbus | Сеть Ethernet | Опросчик Modbus TCP | Станция Modbus TCP |
| | Последовательная шина | Опросчик Modbus TCP | Станция Modbus RTU |

Настройка передачи данных:

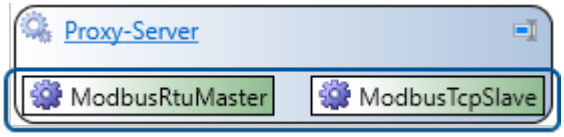
1. Добавьте логический адаптер в источник данных.



2. Добавьте шлюз данных:

2.1. Добавьте **SePlatform.Server**, как описано в разделе 5.1.2. Добавление компонентов домена (стр. 39).

2.2. В **SePlatform.Server** добавьте логические адаптеры для связи с источником данных и с получателем данных.



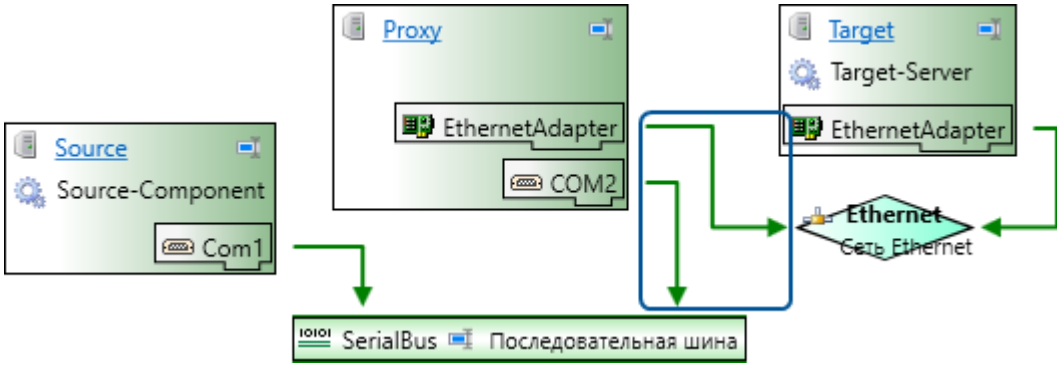

Если используется **Станция Modbus TCP** или **Станция Modbus RTU**:

2.2.1. В свойствах измените номер станции: он должен отличаться от номера станции источника данных (и других станций, доступных получателю данных).

2.2.2. Создайте пустую карту адресов и выберите её в свойствах станции: это нужно, чтобы проект успешно скомпилировался.

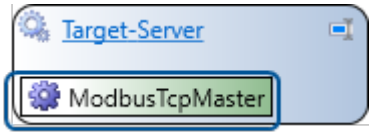
2.3. В **Узел SePlatform.Domain**, где находится шлюз данных, добавьте сетевые адаптеры: для связи с каждой сетью добавьте свой сетевой адаптер.


2.4. Свяжите сетевые адаптеры каждый со своей сетью.

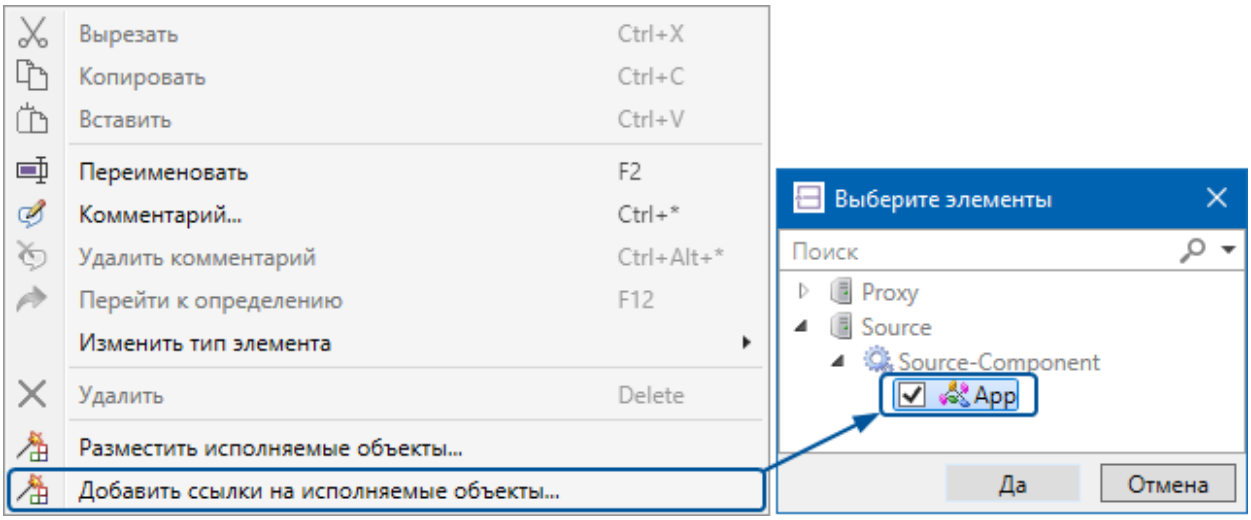


3. В получателе данных:

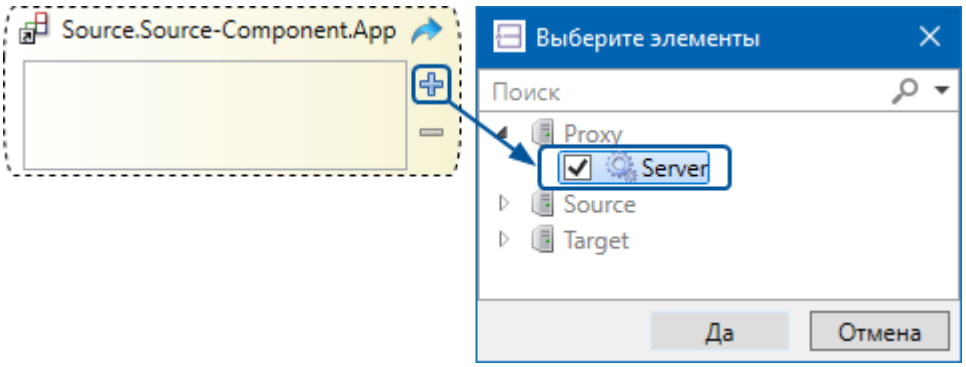
3.1. Добавьте логический адаптер.




3.2. Добавьте ссылку на приложение источника данных.



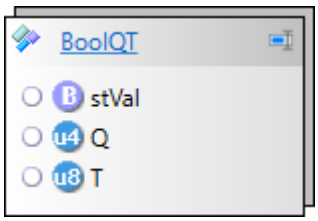

3.3. На изображении ссылки нажмите «+» и выберите **`SePlatform.Server`**, который будет использоваться в качестве шлюза данных.




# 5.13. МЭК 61850: формирование параметра из его атрибутов

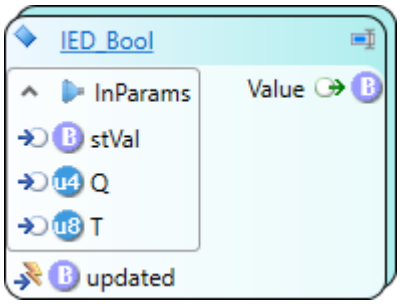
При передаче данных по МЭК 61850 значение, качество и метка времени параметра передаются как значения отдельных сигналов (в МЭК 61850 они называются атрибутами). В примере ниже показано, как при получении этих значений, сформировать из них параметр.

1. Создайте **`Тип сокета`** и добавьте в него элементы типа **`Параметр`**:
   - **`«stVal»`** - тип передаваемого параметра.
   - **`«Q»`** - типа uint4.
   - **`«T»`** - типа uint8.



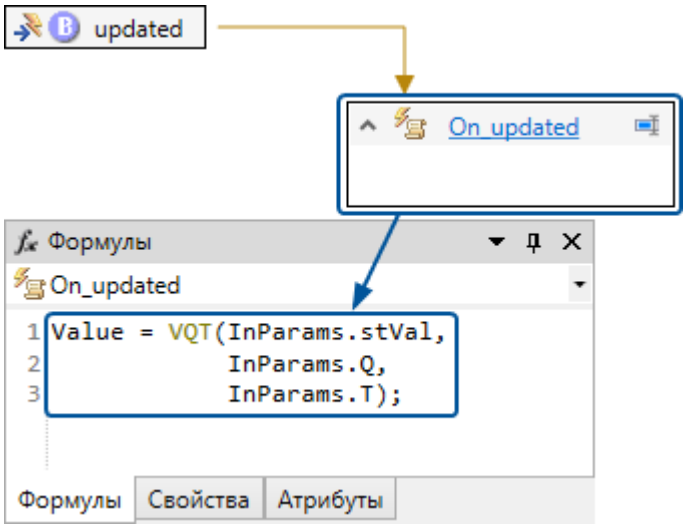

2. Создайте **Логический тип** и добавьте в него следующие элементы:

- входной **Сокет** описанного типа - для записи значений, полученных от IED.
- входное **Событие «updated»** типа bool - признак обновления данных.
- выходной параметр **«Value»** того же типа, что и сигнал **«stVal»** в сокете - в него будет записываться сформированный параметр.

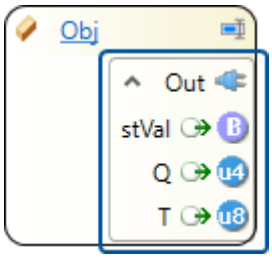



3. Событию **«updated»** добавьте **Обработчик события** (выполняется в контекстном меню) и укажите в нём формулу:

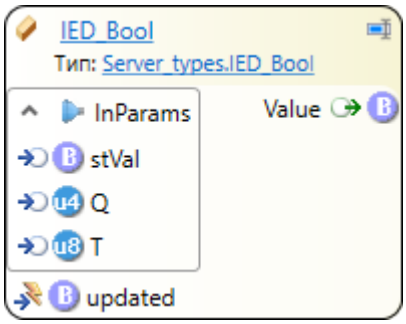
```
Value = VQT(InParams.stVal, InParams.Q, InParams.T);
```



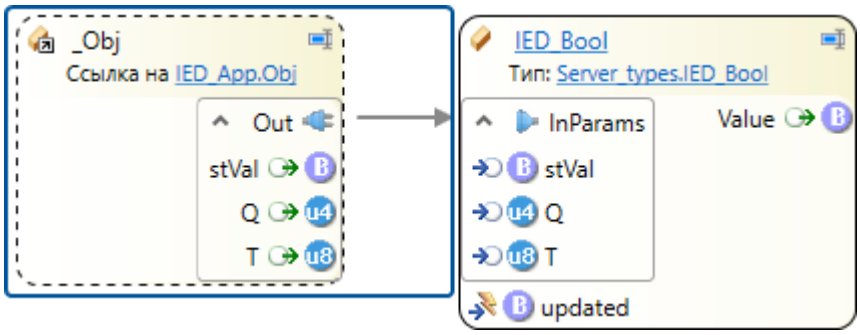

4. В объект, размещённый в IED, добавьте выходной **Сокет** описанного типа.



5. В объект, размещённый в SePlatform.Data Server, добавьте **Логический объект** описанного типа.

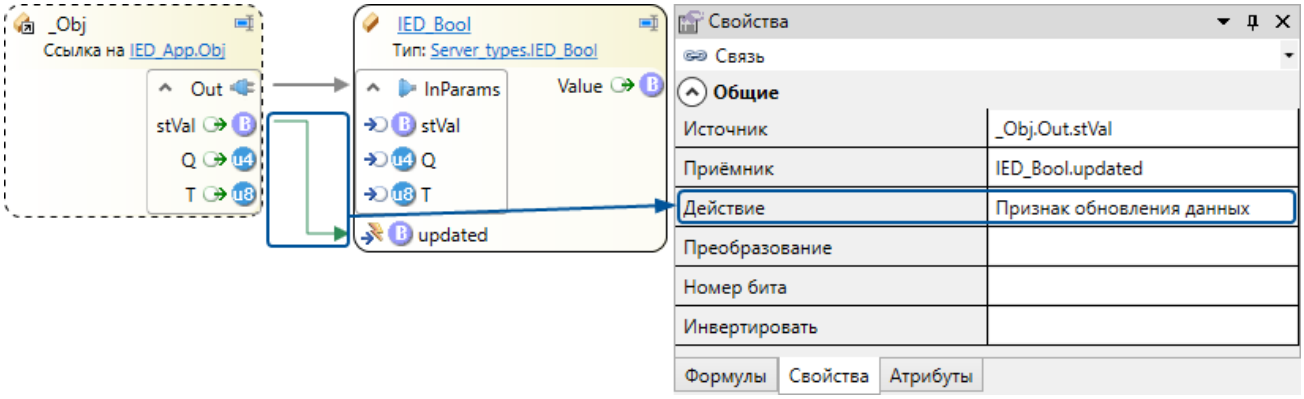



6. Добавьте ссылку на объект в IED и свяжите сокет ссылки с сокетом объекта.




7. Протяните связь от любого параметра в сокете ссылки к событию **«updated»**.

У связи в свойстве **Действие** укажите **«Признак обновления данных»**.




В результате:

- Атрибуты значения, которые передаются из IED в SePlatform.Data Server (значение, качество и метка времени), будут записываться в сигналы сокета.
- Признак обновления данных (индикатор, что все атрибуты значения переданы) будет записываться в сигнал **«updated»**.
- При обновлении сигнала **«updated»** из полученных атрибутов будет формироваться значение сигнала **«Value»**.

# 5.14. Диагностика устройств по SNMP

Протокол SNMP используется для диагностики сетевых устройств и управления их параметрами работы. При использовании протокола SNMP данные передаются между менеджером и агентом.

Менеджер - расположен на компьютере, который выполняет диагностику. Отправляет агентам запросы на получение или изменение параметров, получает от агентов значения запрошенных параметров и

уведомления. В Систэм Платформ роль менеджера выполняет модуль SNMP Manager в составе SePlatform.Data Server.
Агент - расположен на диагностируемом устройстве или компьютере. Принимает запросы от менеджеров, передаёт им значения запрошенных параметров или, если запрос был на изменение значения, изменяет их. При возникновении в устройстве события (критическое значение температуры, изменение состояния порта и пр.) агент отправляет менеджерам уведомление со значением параметра, вызвавшего событие.
Значения параметров, предоставляемые агентом, хранятся в устройстве в виртуальной базе данных MIB (Management Information Base). Список параметров в MIB задаётся разработчиком устройства и не меняется в процессе работы устройства. Параметры в MIB хранятся в древовидной структуре, у каждого параметра есть уникальный OID (Object IDentificator) - его положение в дереве MIB. OID используется в качестве идентификатора параметра в запросах и в уведомлениях.

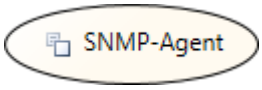
## 5.14.1. Настройка диагностики устройств

ПРИМЕЧАНИЕ

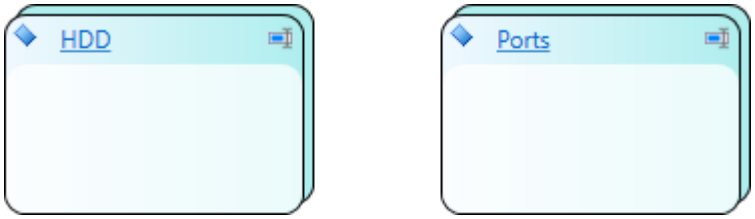
В дистрибутив SePlatform.Development Studio входит пример решения, в котором настроена диагностика устройств по SNMP: `Пуск → SePlatform → Пример решения с SNMP`.

Ниже описано, как настроить в проекте диагностику устройств по SNMP. В сценарии используются шаги и этапы, описанные в разделах 5.1. Описание домена (стр. 36) и 5.2. Добавление объектов (стр. 56); для шагов приведены ссылки на топики, в которых описано, как выполнять нужное действие.

1. Подготовьте типы для агентов:
   1.1. Создайте аспект - **«SNMP-Agent»** (см. раздел 5.2.1. Создание аспектов (стр. 57)).

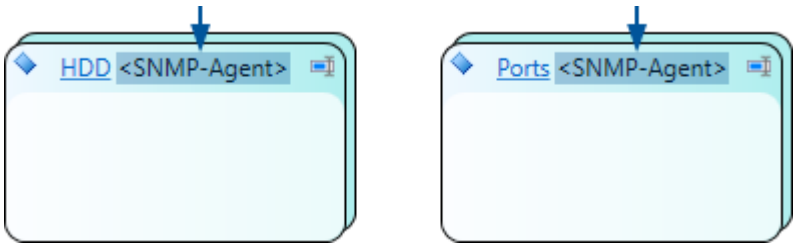



   1.2. Опишите типы диагностируемых объектов в составе устройств: **«HDD»**, **«RAM»**, **«CPU»** и пр.

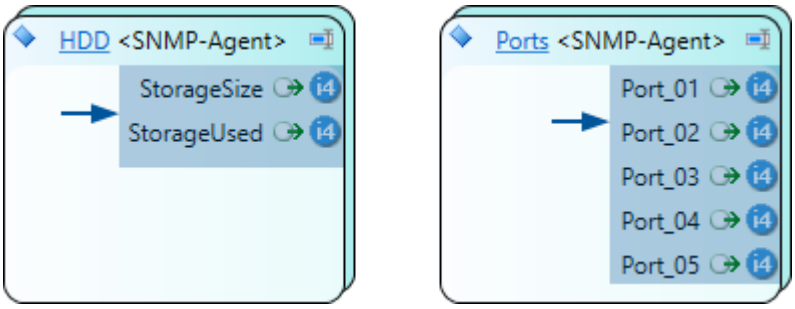



Для каждого типа:

1.2.1. Укажите аспект - **«SNMP-Agent»**.

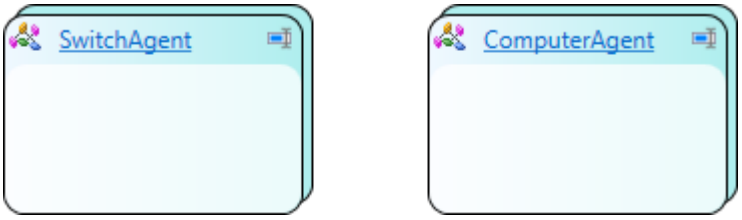


1.2.2. Добавьте сигналы - параметры, которые будут передаваться по SNMP.



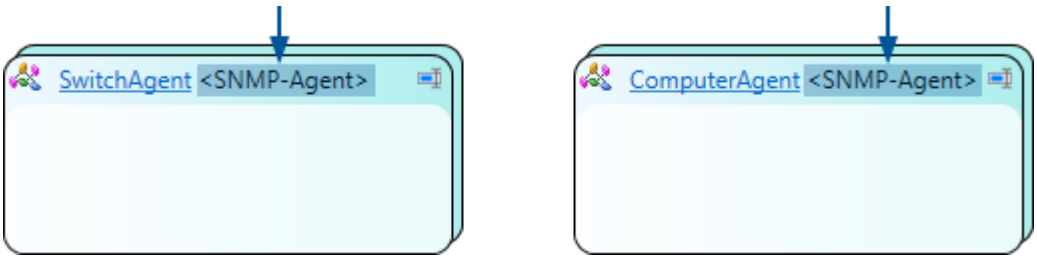

Описание типов, включая указание аспекта и добавление сигналов, приведено в разделе 5.2.2. Описание объектов (стр. 58).

1.3. Опишите типы приложений (см. раздел 5.1.3. Добавление приложений (стр. 44)). Для каждого типа устройств опишите отдельный тип приложения; если есть однотипные устройства, для них опишите один тип приложения.

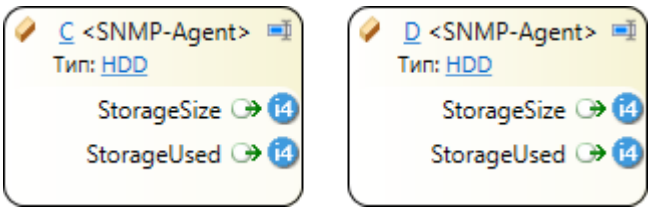



Для каждого типа приложения:

1.3.1. Укажите аспект - **«SNMP-Agent»**.



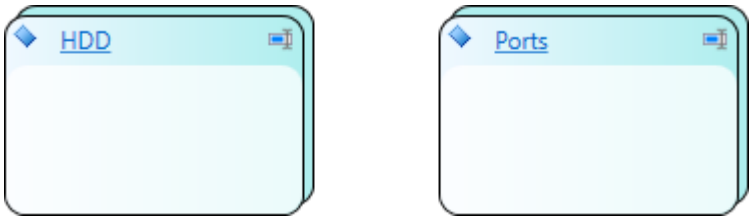

1.3.2. Добавьте в него объекты описанных типов. Набор объектов зависит от конфигурации устройства: для одного типа устройств потребуется добавить два объекта типа **«HDD»**, для другого - ни одного.




2. Для описанных типов опишите их представления в аспекте сервера:

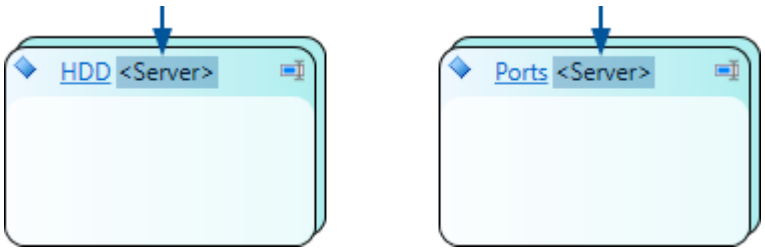
2.1. Создайте аспект - **«Server»** (не нужно, если в проекте уже есть аспект для размещения объектов в сервере).

2.2. Для каждого типа объекта в аспекте **«SNMP-Agent»** опишите соответствующий тип в аспекте **«Server»**.

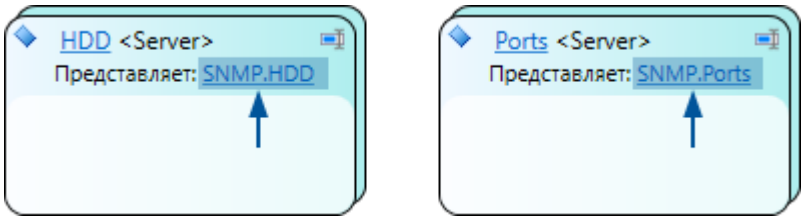



Для каждого типа:

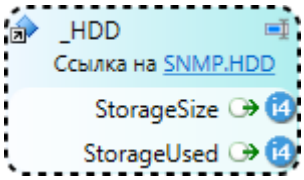
2.2.1. Укажите аспект - **«Server»**.



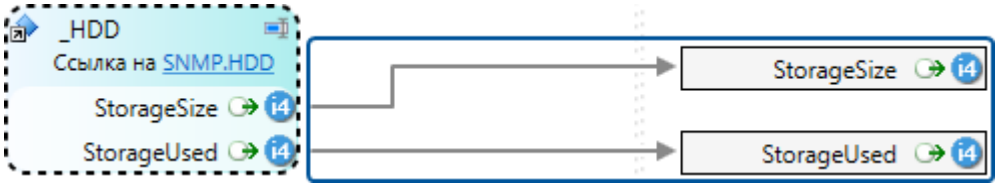

2.2.2. Укажите представляемый тип - соответствующий тип в аспекте **«SNMP-Agent»**.



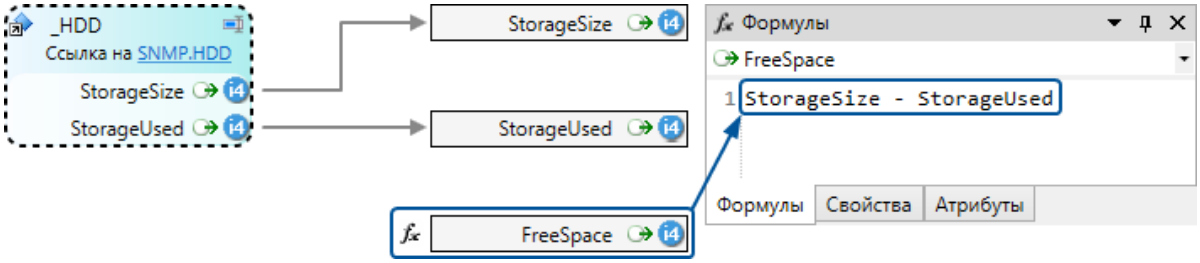

2.2.3. Добавьте в тип аспектную ссылку на соответствующий тип в аспекте **«SNMP-Agent»**.




2.2.4. Экспонируйте сигналы ссылки.

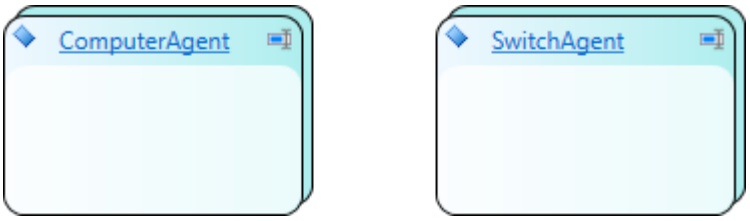



2.2.5. При необходимости, добавьте сигналы, значения которых будут вычисляться на основе полученных значений. С помощью формул или обработчиков событий опишите правила вычисления значений таких сигналов.



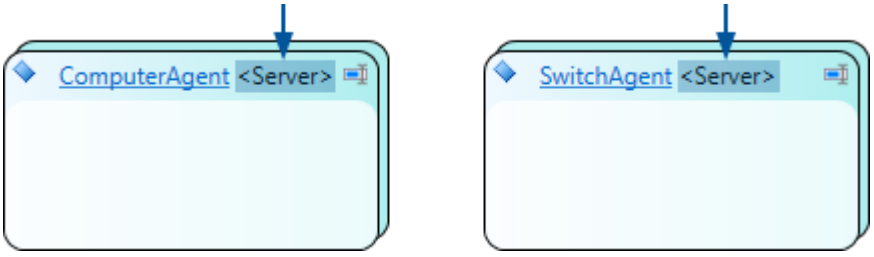

Описание типов, включая указание аспекта, добавление аспектных ссылок, экспонирование сигналов и добавление вычислений, описано в разделе 5.2.2. Описание объектов (стр. 58).

2.3. Для каждого типа приложения в аспекте **«SNMP-Agent»** опишите тип объекта в аспекте **«Server»**: объекты этих типов в дальнейшем будут размещены в приложении сервера для обмена данными с агентами.

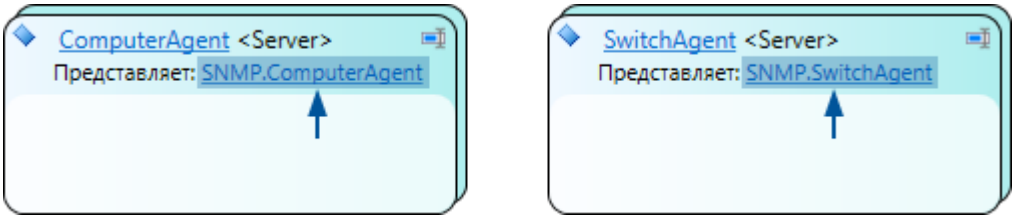



Для каждого типа:

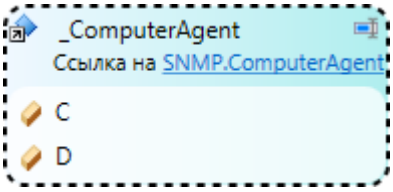
2.3.1. Укажите аспект - **«Server»**.



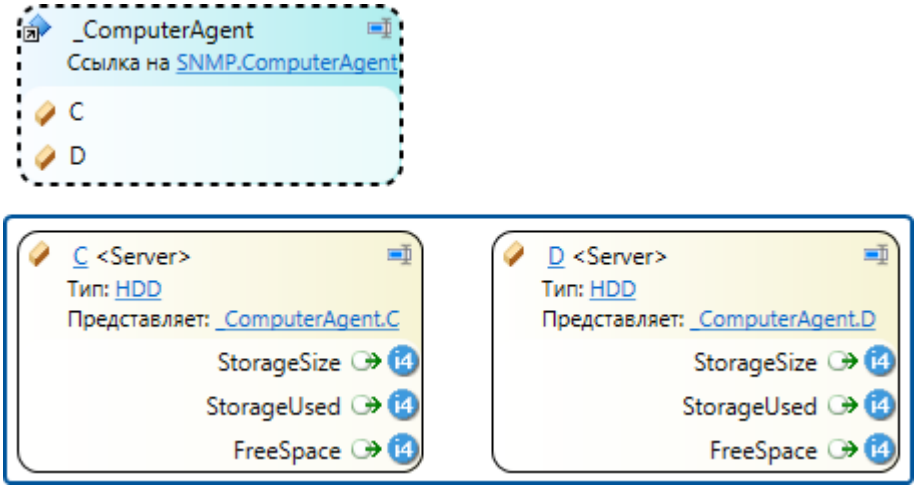

2.3.2. Укажите представляемый тип - приложение в аспекте **«SNMP-Agent»**.




2.3.3. Добавьте в тип аспектную ссылку на приложение в аспекте **«SNMP-Agent»**.

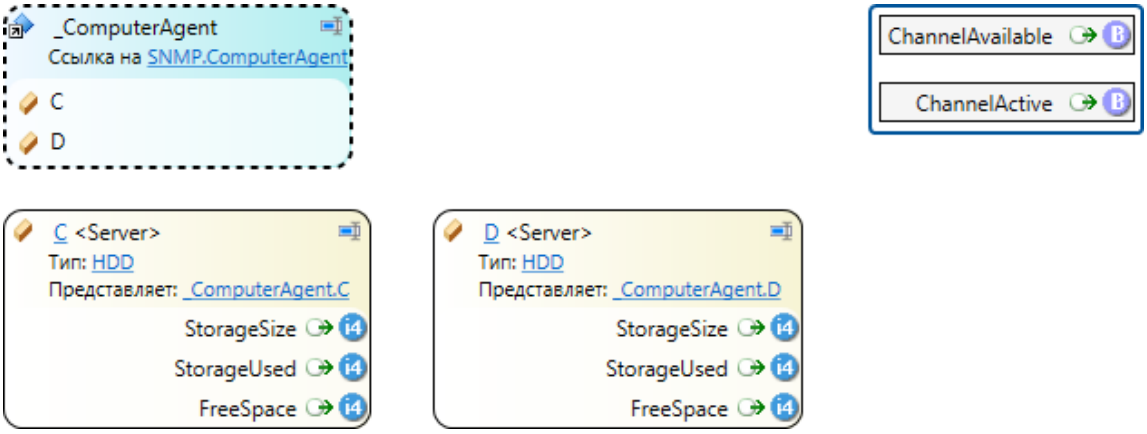



2.3.4. Разместите в типе представления для объектов, находящихся в типе приложения, на которое указывает ссылка. Для размещения используйте мастер создания представлений, как описано в разделе 5.2.3. Размещение объектов (стр. 84).

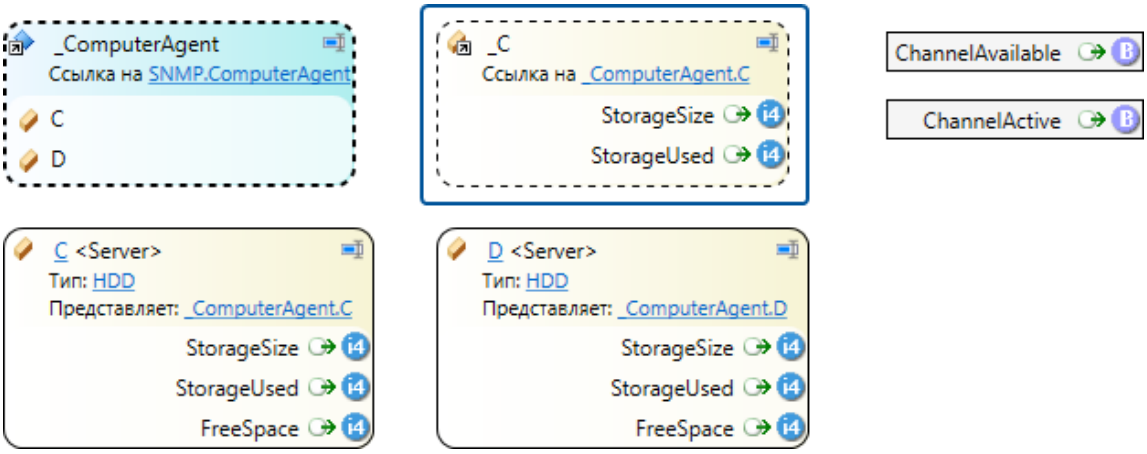


2.3.5. Добавьте в тип сигналы для диагностики связи с агентом:

2.3.5.1. Добавьте два сигнала типа bool: **«ChannelAvailable»** и **«ChannelActive»**.

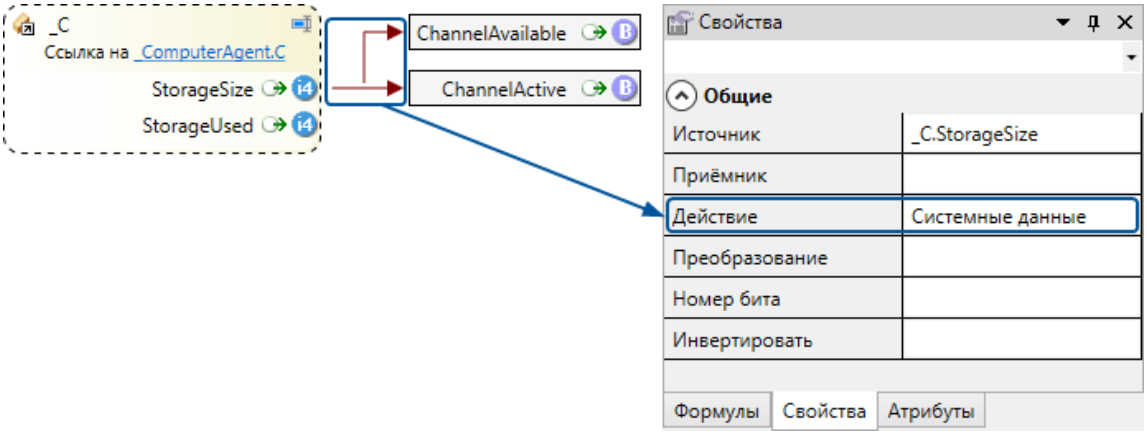



2.3.5.2. Добавьте в тип ссылку на любой объект в типе приложения. Ссылка нужна для создания связи с добавленными сигналами.



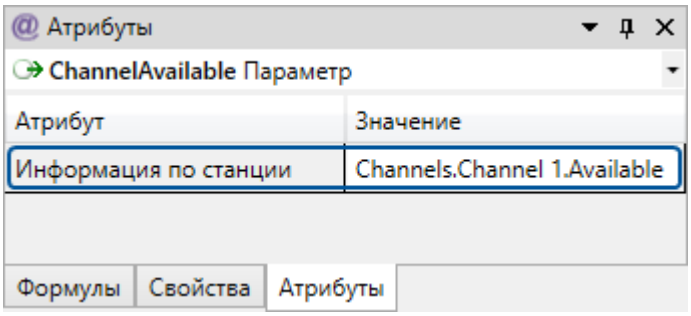

2.3.5.3. От любого сигнала в ссылке протяните связи к добавленным сигналам.

У связей в свойстве **Действие** выберите **«Системные данные»**.

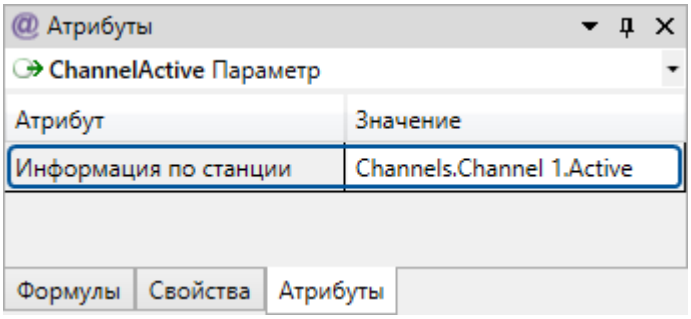



Данный вид связи описан в разделе 5.2.2.3. Добавление связей (стр. 69).

2.3.5.4. Сигналу **«ChannelAvailable»** добавьте атрибут **Информация по станции** и укажите ему значение **«Channels.Channel 1. Available»** (сервисный сигнал в модуле SNMP Manager).




2.3.5.5. Сигналу **«ChannelActive»** добавьте атрибут **Информация по станции** и укажите ему значение **«Channels.Channel 1.Active»** (сервисный сигнал в модуле SNMP Manager).

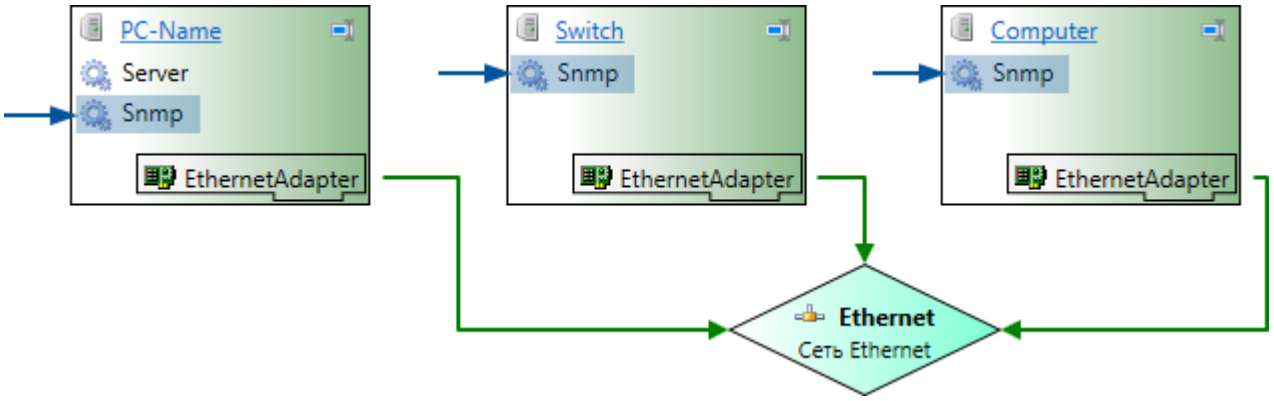



В результате в сигналы будут записываться значения сервисных сигналов модуля SNMP Manager, определяющих доступность и активность канала связи с агентом.

ПРИМЕЧАНИЕ

Сервисные сигналы приведены в документации на модуль SNMP Manager.

3. Добавьте агенты в домен.

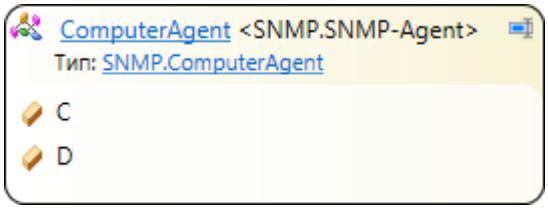



Чтобы добавить агента:

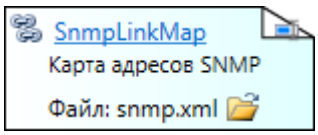
3.1. В домен добавьте исполняющий компонент. Исполняющий компонент нужно добавить в компьютер/устройство, на котором будет расположен агент в среде исполнения.


Добавление исполняющих компонентов описано в разделе 5.1.2. Добавление компонентов домена (стр. 39).

3.2. В исполняющий компонент добавьте приложение одного из описанных типов в аспекте **«SNMP-Agent»**.




3.3. В приложение добавьте карту адресов SNMP. Добавление карты адресов описано в разделе 5.1.4. Настройка передачи данных (стр. 48).




ПРИМЕЧАНИЕ

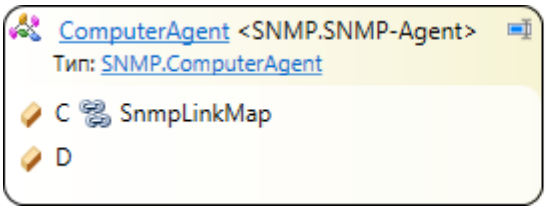
Для однотипных устройств используйте один файл адресов: создайте его для одного устройства, а для других устройств выберите созданный файл. В этом случае указывать адреса сигналов потребуется только для одного устройства каждого типа: при создании карты адресов.

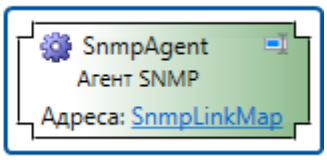
3.4. В карте адресов для каждого сигнала укажите:

| Поле | Значение |
|---|---|
| **Привязка** | Укажите **«непосредственно»** - параметры адреса заданы в данной карте адресов. |
| **OID элемента** | Укажите OID параметра. OID можно узнать в документации на устройство или извлечь из MIB устройства с помощью утилит. |
| **Индекс элемента массива** | Индекс элемента в массиве. Указывается только если OID указывает на массив. |
| **Композитный тип объекта** | Указывается только для сложных типов данных:<br>› **«Дата и время»**<br>› **«Физический адрес»**<br>› **«IP-адрес»**<br>Для всех остальных оставьте поле пустым. |
| **Способ получения данных** | Укажите способ получения данных или оставьте пустым:<br>› **«Только через опрос»**<br>› **«Только через уведомления»**<br>› **«Через опрос и уведомления»**<br>› Пусто - равносильно значению **«Через опрос и уведомления»** |

3.5. В исполняющий компонент добавьте логический адаптер **Агент SNMP**.

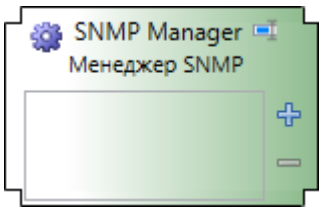
В свойствах элемента укажите добавленную карту адресов и параметры агента: порты для опроса и уведомлений, версию протокола и прочие.



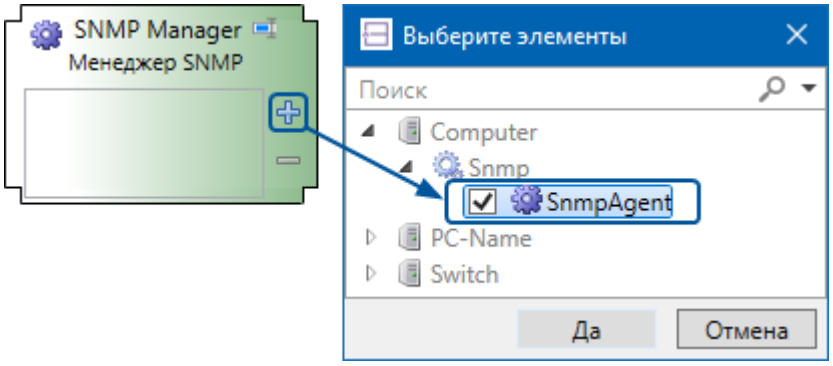
4. В SePlatform.Data Server настройте обмен данными с агентами:

4.1. Добавьте логический адаптер **Менеджер SNMP** для получения данных от агентов.



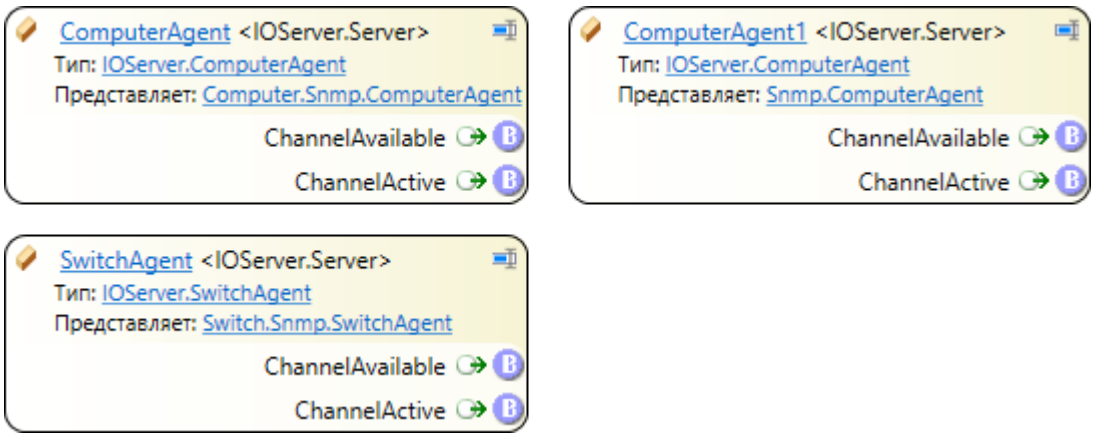

Логический адаптер будет опрашивать все агенты, находящиеся в той же сети, и получать от них уведомления.

4.2. (Опционально) На изображении логического адаптера нажмите «+» и выберите один или несколько агентов.




Для каждого выбранного агента в логический адаптер будет добавлен элемент **Параметры взаимодействия с агентом**. В свойствах элемента укажите параметры связи с агентом: период опроса и другие.

4.3. В приложении разместите объекты - представления приложений агентов. Для размещения используйте мастер создания представлений (см. раздел 5.2.3. Размещение объектов (стр. 84)).




В результате полученные от агентов значения будут записываться в сигналы размещённых объектов.

# 5.15. Повышение надёжности проекта автоматизации

Чтобы повысить надёжность проекта автоматизации и защитить его от выхода из строя компонентов, на которых он исполняется, нужно добавить резервные компоненты. Резервный компонент работает в паре с основным компонентом и размещается на другом компьютере или устройстве; при прекращении работы основного компонента, резервный компонент продолжит работу и будет выполнять его функции в проекте.

## 5.15.1. Виды резервирования

Есть два вида резервирования:

- Дублирование
- Горячее резервирование

### 5.15.1.1. Дублирование

При дублировании основной и резервный компоненты функционируют одновременно: получают, обрабатывают и передают данные. Клиенты (компоненты, которые к ним подключаются) сами выбирают, к какому компоненту подключиться и от которого получать данные.

Преимущество дублирования:

- При прекращении работы одного из компонентов остановки не происходит, система продолжает работать.

Ограничение на применение дублирования:

- Дублирование неприменимо, если в системе недопустима одновременная работа основного и резервного компонента:
  - Используются контроллеры с ограниченным количеством одновременных подключений.
  - Существуют ограничения на количество параллельных транзакций.
  - Используются протоколы, в которых невозможно применение нескольких опросчиков одновременно.

### 5.15.1.2. Горячее резервирование

При горячем резервировании основной и резервный компоненты образуют резервную пару. В резервной паре основной компонент работает в полнофункциональном режиме, а резервный компонент находится в режиме ожидания. При прекращении работы основного компонента автоматически выполнится резервный переход: резервный компонент переключится в полнофункциональный режим и будет выполнять функции основного компонента. Клиенты в процессе работы подключаются к тому компоненту резервной пары, который является основным.

Преимущества горячего резервирования:

- Резервный компонент начинает работать автоматически при потере связи с основным компонентом.
- Горячее резервирование применимо, если недопустима одновременная работа основного и резервного компонентов.

Ограничение на применение горячего резервирования:

- Резервный переход может занять до нескольких секунд. Поэтому горячее резервирование неприменимо, если недопустимо даже краткосрочное прекращение работы компонента.

## 5.15.2. Резервирование компонентов

Ниже описано, каким образом в проекте автоматизации резервируются различные компоненты. Для SePlatform.AccessPoint описано резервирование АРМов, на которых они работают.

ПРИМЕЧАНИЕ

Добавление компонентов, для которых выполняется резервирование, описано в разделе 5.1.2. Добавление компонентов домена (стр. 39).

## 5.15.2.1. Резервирование SePlatform.Data Server

Для SePlatform.Data Server доступны оба вида резервирования: дублирование и горячее резервирование. Для основного и резервного SePlatform.Data Server создаётся один файл конфигурации, который при развёртывании применяется к обоим экземплярам.

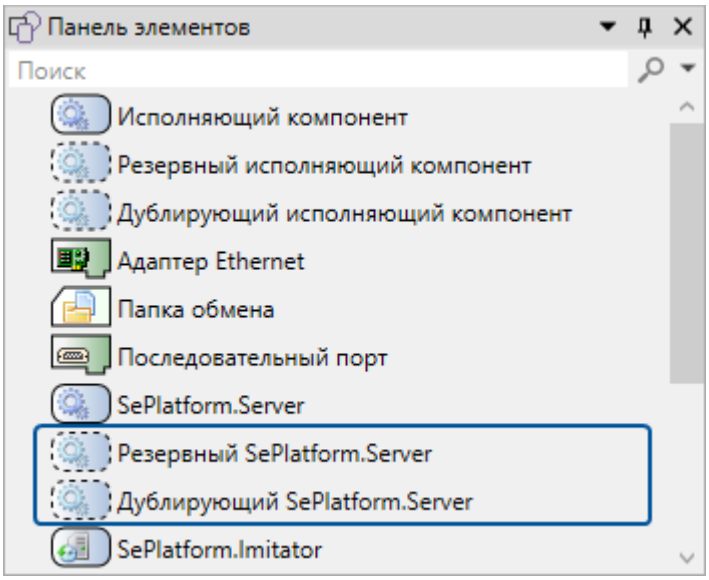
ОБРАТИТЕ ВНИМАНИЕ

Чтобы резервный SePlatform.Data Server можно было конфигурировать при развёртывании, его нужно добавить в домен (выполняется в компоненте SePlatform.Domain так же, как добавление основного SePlatform.Data Server).

Чтобы добавить резервный SePlatform.Data Server:

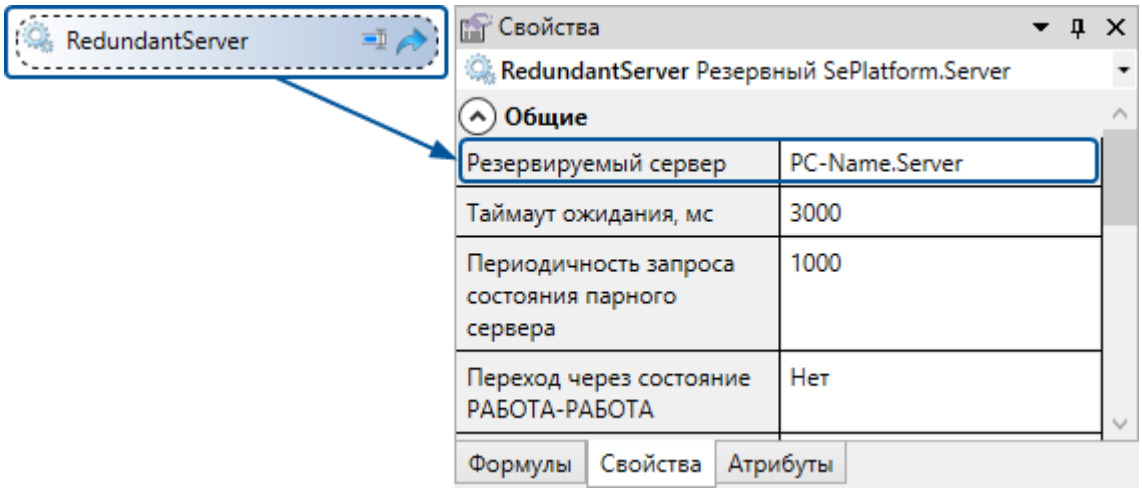
1. Перейдите в **`SePlatform.Domain`**.
2. Добавьте **`Узел SePlatform.Domain`** - компьютер, на котором расположен резервный SePlatform.Data Server.

   В качестве имени элемента укажите сетевое имя компьютера.
3. Перейдите в добавленный элемент.
4. В зависимости от вида резервирования добавьте:
   - **`Резервный SePlatform.Server`** - работает в режиме горячего резервирования.
   - **`Дублирующий SePlatform.Server`** - работает в режиме дублирования.



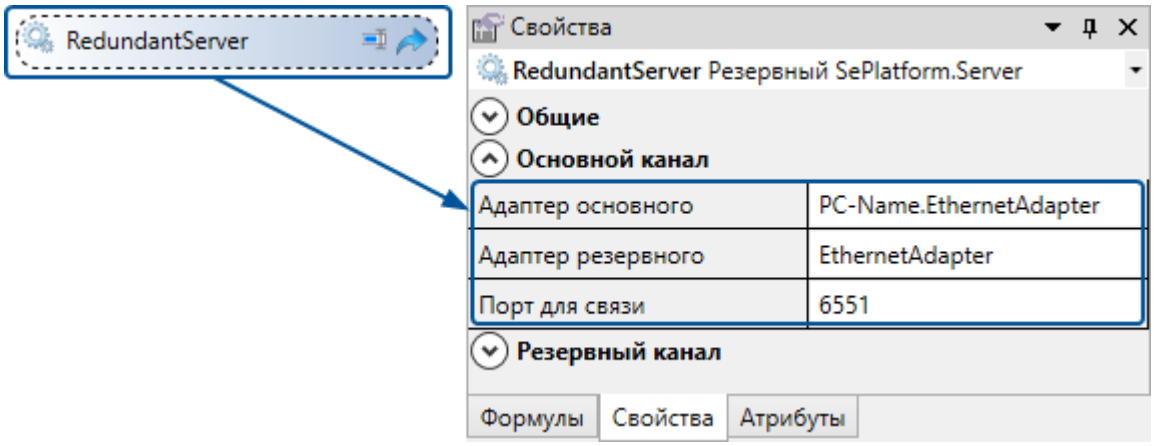

Имя элемента должно совпадать с именем, настроенным в SePlatform.Domain.

5. В свойствах элемента укажите основной сервер.




6. Если добавлен **Резервный SePlatform.Server**, в свойствах укажите его параметры связи с основным сервером:

- Параметры основного канала: порт и адаптеры, используемые для связи (адаптеры добавляются при описании интерфейсов передачи данных (стр. 48)).
- (При наличии) Параметры резервного канала: порт и адаптеры, используемые для связи по резервному каналу.




## 5.15.2.2. Резервирование АРМов

Для резервирования АРМов используется дублирование: в проект автоматизации добавляются несколько АРМов, которые получают одинаковые данные и выполняют одни и те же функции.
В проекте SePlatform.Development Studio резервные АРМы описывать не требуется: одному элементу **Рабочее место**, могут соответствовать любое количество АРМов. Однако для того, чтобы при развёртывании можно было конфигурировать SePlatform.AccessPoint в составе АРМа, нужно добавить SePlatform.AccessPoint в составе каждого АРМа в домен (выполняется в компоненте SePlatform.Domain).

## 5.15.2.3. Резервирование серверов истории

Для резервирования серверов истории используется дублирование: в проект добавляются несколько серверов истории. Резервные серверы истории добавляются так же, как основной сервер истории: см. раздел 5.1.2. Добавление компонентов домена (стр. 39).
После добавления серверов истории, нужно настроить сохранение в них данных: в экземплярах SePlatform.Data Server в элементах **Модуль истории** и **OPC AE Сервер** указать базы данных в составе

серверов истории. Данные будут сохраняться во все указанные модулям базы данных; для запроса сохранённых данных будет использоваться та БД, с которой быстрее всего установлено соединение.

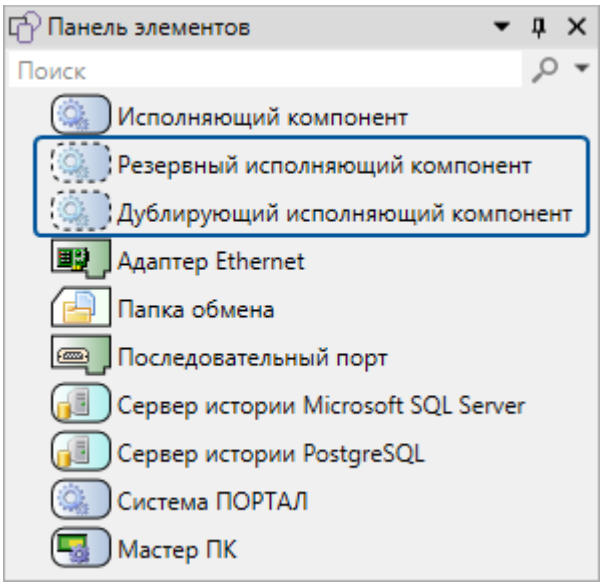
### 5.15.2.4. Резервирование исполняющих компонентов

Для исполняющих компонентов доступны оба вида резервирования: дублирование и горячее резервирование. Информация о резервных исполняющих компонентах используется при создании конфигураций для SePlatform.Data Server и SePlatform.AccessPoint.

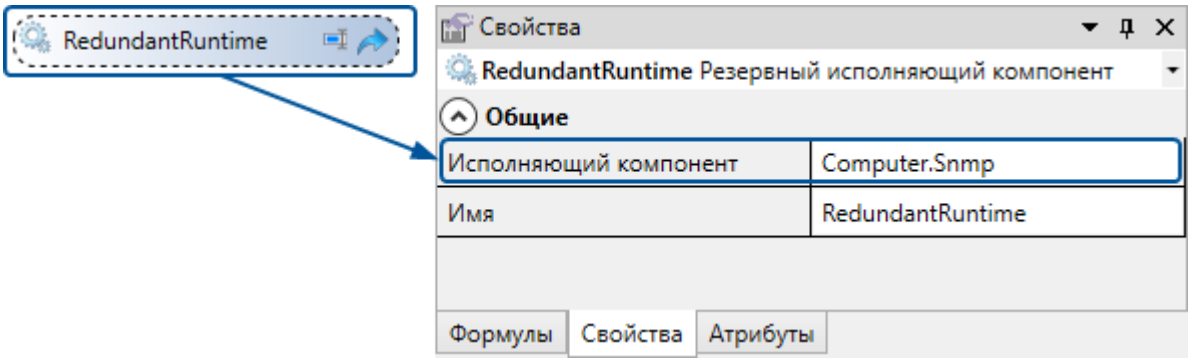
Чтобы добавить резервный исполняющий компонент:

1. Перейдите в **`SePlatform.Domain`**.
2. Добавьте **Компьютер**, на котором расположен резервный исполняющий компонент.

   В качестве имени элемента укажите сетевое имя компьютера или устройства.
3. Перейдите в добавленный элемент.
4. В зависимости от вида резервирования добавьте:
   - **Резервный исполняющий компонент** - работает в режиме горячего резервирования
   - **Дублирующий исполняющий компонент** - работает в режиме дублирования




5. В свойствах элемента укажите исполняющий компонент, который он резервирует.




# 5.16. Импорт/экспорт элементов

Элементы можно экспортировать в файл и импортировать из файла. Файл `*.omx-export` имеет xml-формат содержимого и может быть отредактирован (или создан) в другой программе.

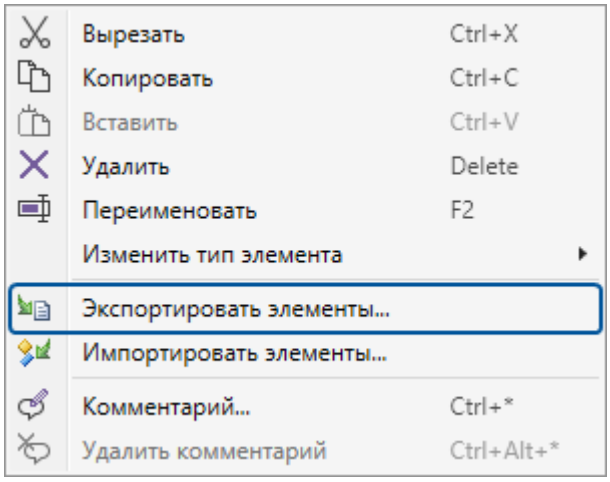
Импорт/экспорт элементов используется для:

- добавления в проект элементов, описанных в другой программе.
- редактирования элементов в другой программе.

## 5.16.1. Экспорт элементов

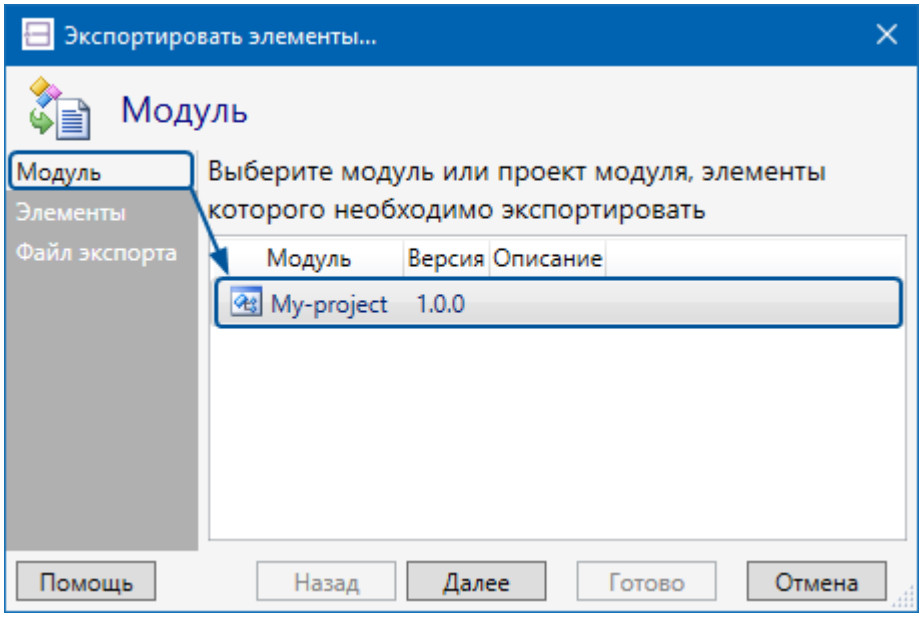
Чтобы экспортировать элементы:

1. В обозревателе решений в контекстном меню решения, проекта или элемента выберите **Экспортировать элементы...**.

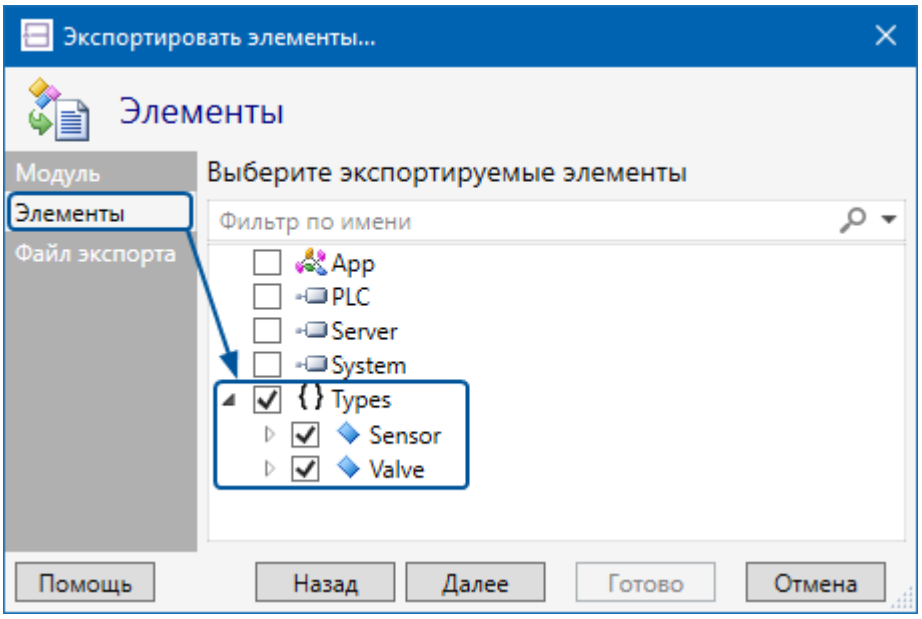



Откроется мастер экспорта.

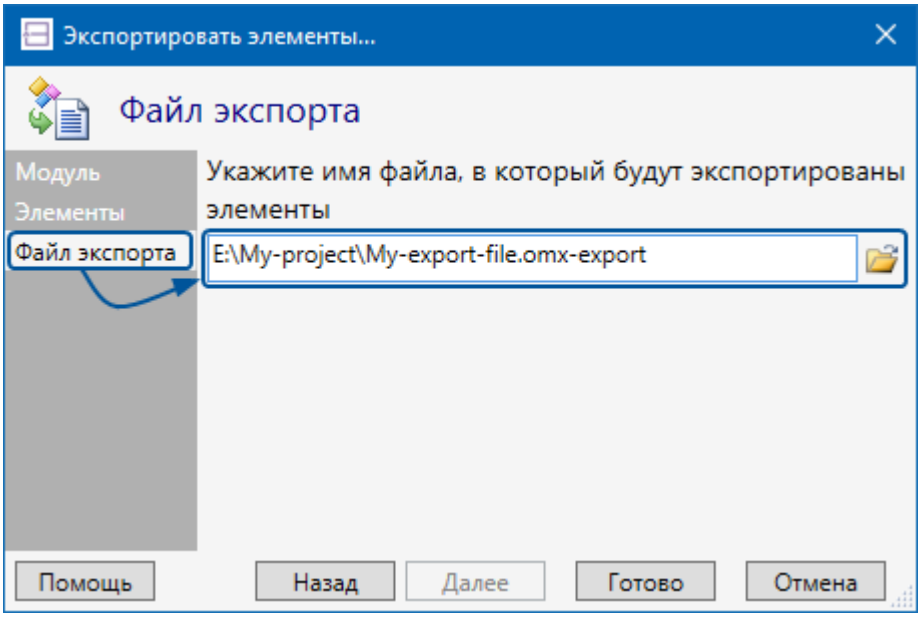
2. (Если импорт вызван для решения) Выберите проект.



3. Выберите элементы.




4. (Опционально) В плане экспорта посмотрите, какие элементы будут экспортированы в файл.
5. Укажите файл.




6. Нажмите **Готово**.

В результате:

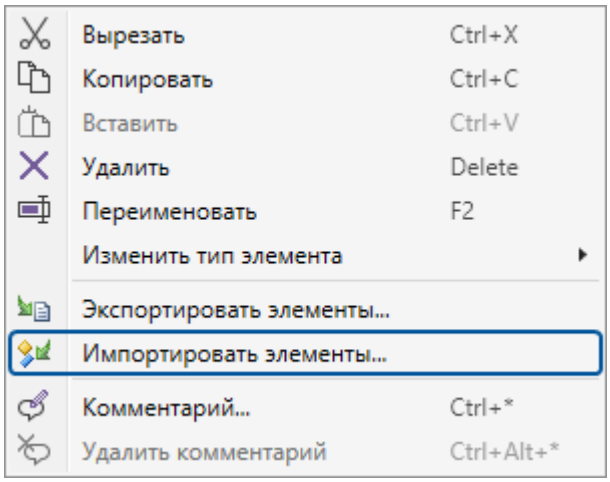
- Будет создан указанный файл. Если файл существует, он будет перезаписан.
- В файле будут сохранены выбранные элементы. Для каждого элемента сохраняются:
  - свойства
  - атрибуты
  - вложенные элементы

## 5.16.2. Импорт элементов

Чтобы импортировать элементы:

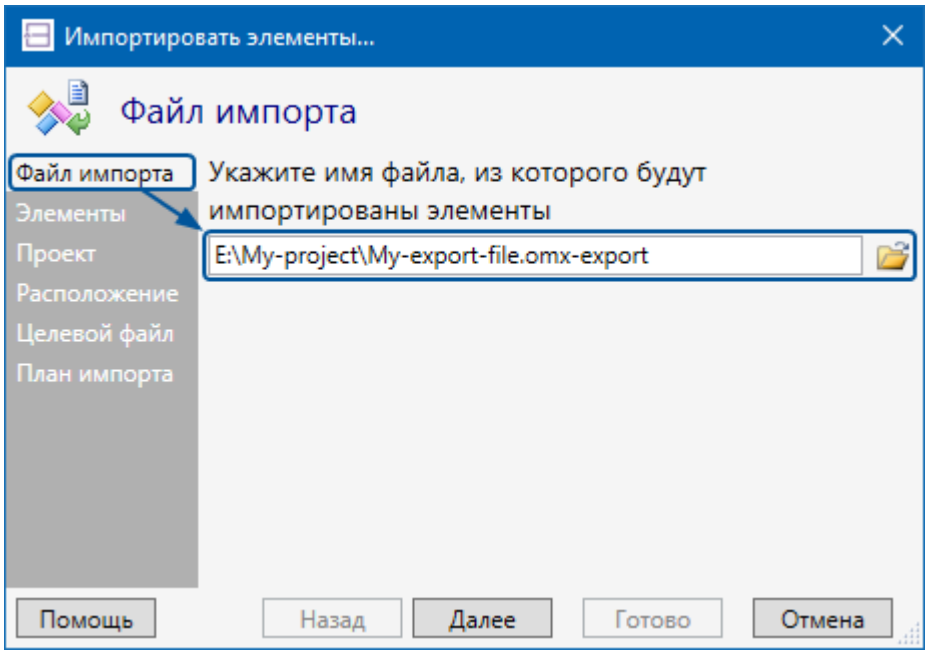
1. В обозревателе решений в контекстном меню решения, проекта или элемента выберите

**Импортировать элементы...**.

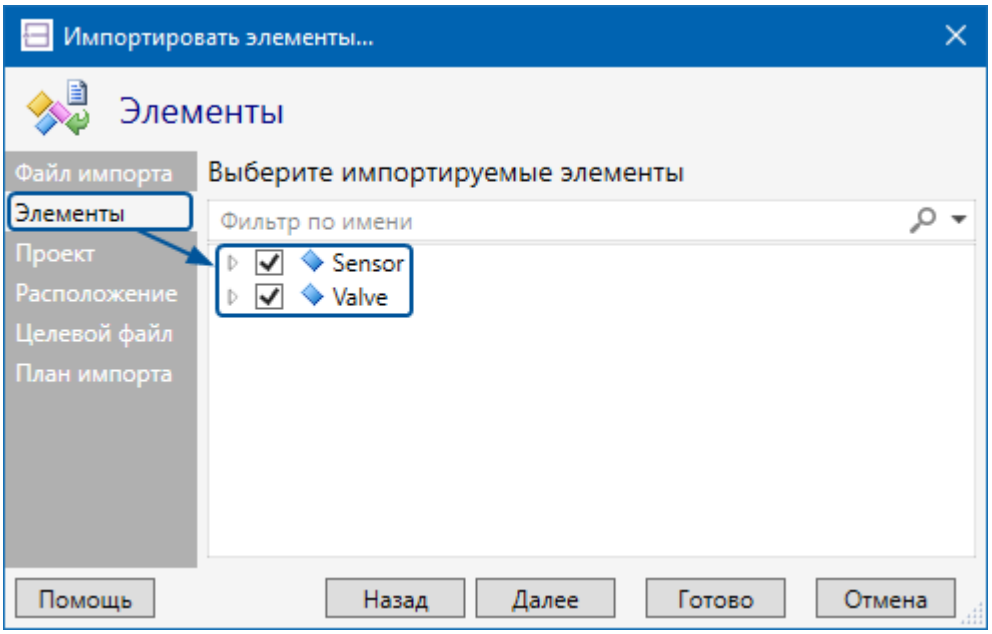



Откроется мастер импорта.

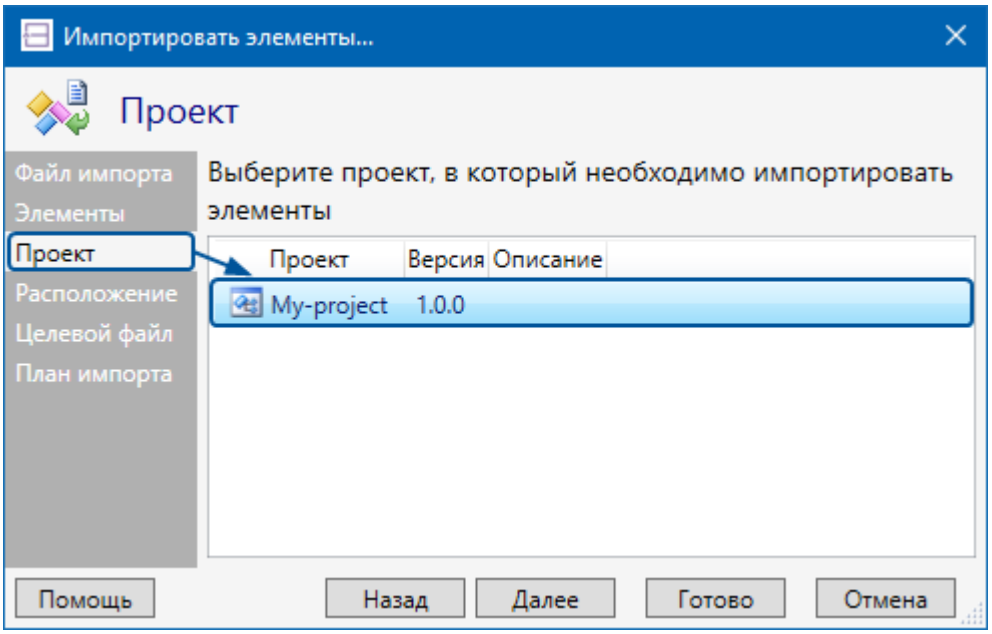
2. Выберите файл.



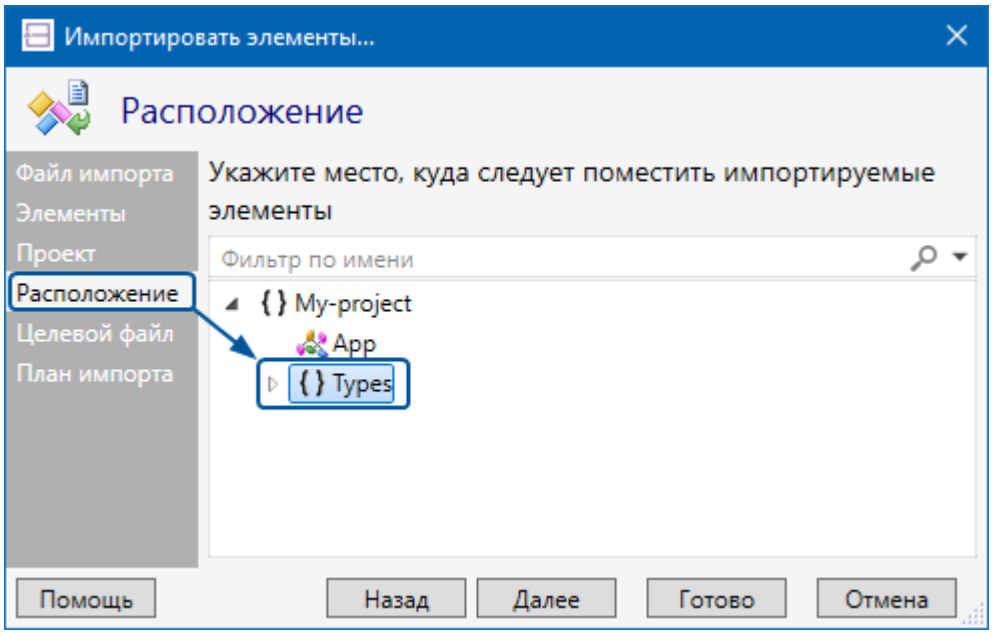
3. Выберите элементы в файле, которые нужно импортировать.



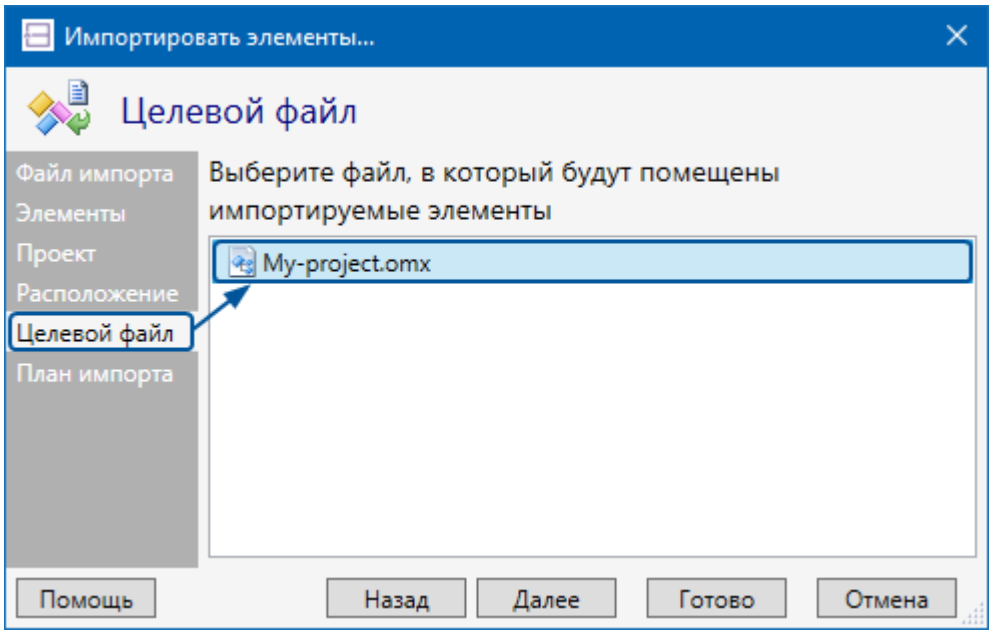

4. (Если импорт вызван для решения) Выберите проект.



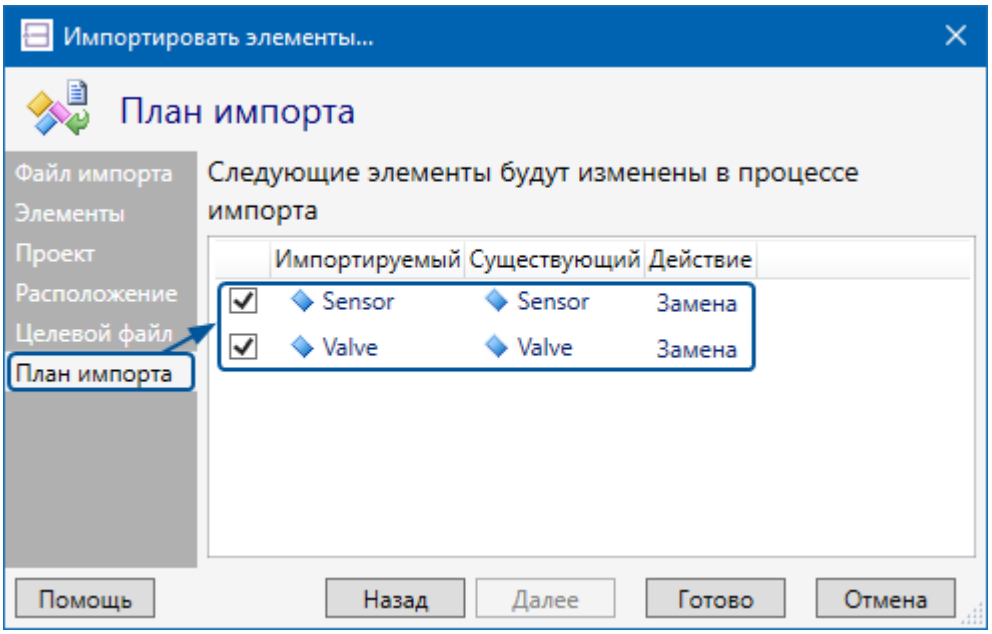
5. Укажите место в проекте, куда следует импортировать элементы.




6. Выберите файл проекта, в который будет записана информация об импортируемых элементах.



7. (Опционально) В плане импорта посмотрите изменения, которые будут внесены в проект при импорте.




При необходимости, снимите флаги с тех изменений, которые вносить не нужно.

8. Нажмите **Готово**.

В результате выбранные элементы будут импортированы в выбранный проект и размещены в указанном месте. При импорте элемента будет выполнено следующее:

- Если элемент отсутствует в проекте - он будет добавлен в указанное место.

Наличие элемента проверяется по его идентификатору - атрибут **uuid** у элемента в файле. Если идентификатор не указан, он будет сгенерирован при добавлении элемента.

> ПРИМЕЧАНИЕ
> Чтобы узнать идентификатор элемента, экспортируйте его: идентификатор будет указан в файле экспорта.

- Если элемент есть в проекте - он будет заменён на элемент, описанный в файле, и перемещён в указанное место.

При замене элемента будет заменено всё его описание, включая вложенные элементы.

- Если в файле элементу добавлен атрибут удаления **«isDeleted="true"»** - при импорте элемент будет удалён из проекта.

Если удаляется объект - будут также удалены объекты, представляющие его. Если удаляется тип - будут также удалены типы, представляющие его, и объекты всех удалённых типов.

> ПРИМЕЧАНИЕ
> Атрибут удаления используется, если элементы разрабатываются в другой программе и неоднократно импортируются в проект.
>
> В этом случае, если в процессе разработки элемент был удалён, в файле следует оставить описание элемента, но добавить ему атрибут удаления, чтобы при импорте элемент был удалён из проекта.

ОБРАТИТЕ ВНИМАНИЕ

Описанные правила импорта не действуют на вложенные элементы. Вложенные элементы являются содержимым элемента, в который они вложены, и полностью заменяются при импорте этого элемента.

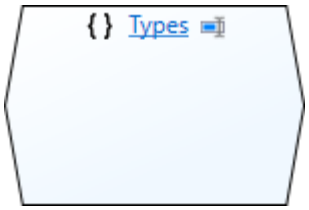
## 5.16.3. Демонстрационный пример импорта/экспорта

В этом примере мы:

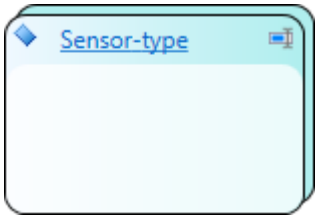
- экспортируем элементы из проекта
- в файле экспорта изменим описание элементов
- импортируем изменённый файл в проект

### Подготовка элементов

1. В новом проекте добавляем **Пространство имён** с именем **«Types»** и переходим в него.



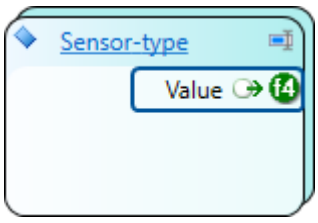

2. Добавляем **Логический тип** с именем **«Sensor-type»** и переходим в него.



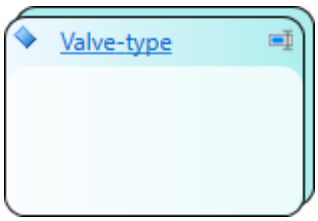

3. Добавляем **Параметр** с именем **«Value»**.

Свойства:

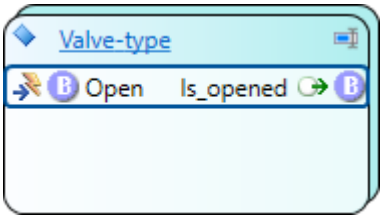
- **Тип - «float»**
- **Направление - «Выход»**



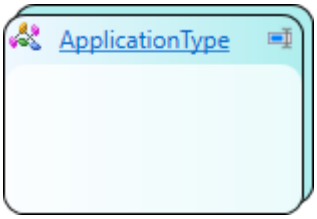

4. В пространство имён **«Types»** добавляем **Логический тип** с именем **«Valve-type»** и переходим в него.



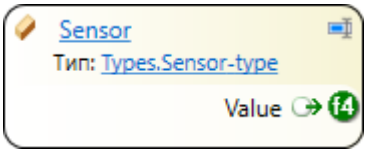
5. Добавляем сигналы:
   - **Событие** с именем **«Open»**:
     - **Тип** - **«bool»**
     - **Направление** - **«Вход»**
   - **Параметр** с именем **«Is_opened»**:
     - **Тип** - **«bool»**
     - **Направление** - **«Выход»**



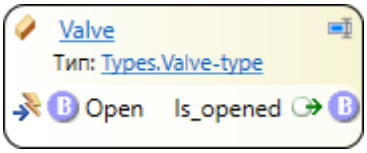

6. Переходим в **Содержимое модуля** (корень проекта), добавляем **Тип приложения** и переходим в него.



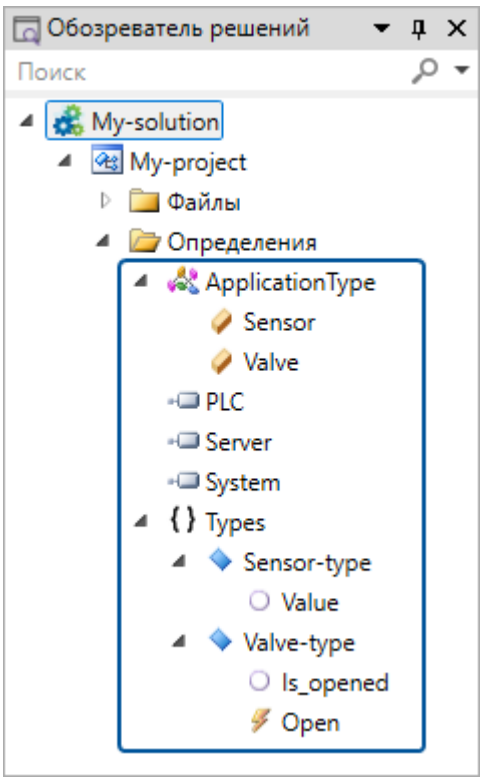

7. Добавляем **Логический объект** с именем **«Sensor»** типа **«Sensor-type»**.




8. Добавляем **Логический объект** с именем **«Valve»** типа **«Valve-type»**.

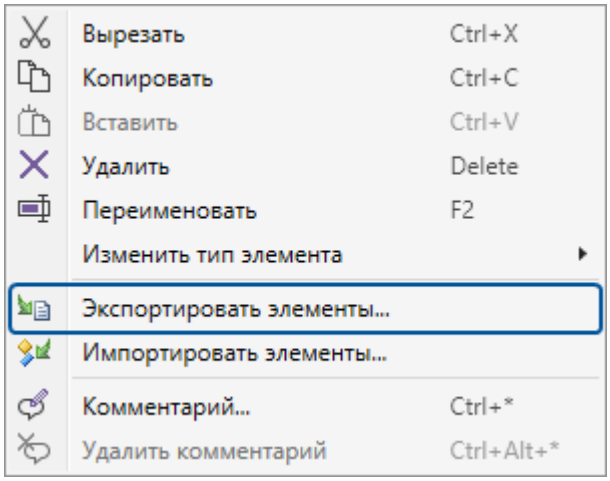



Получившееся дерево решения будет иметь вид как на рисунке ниже.

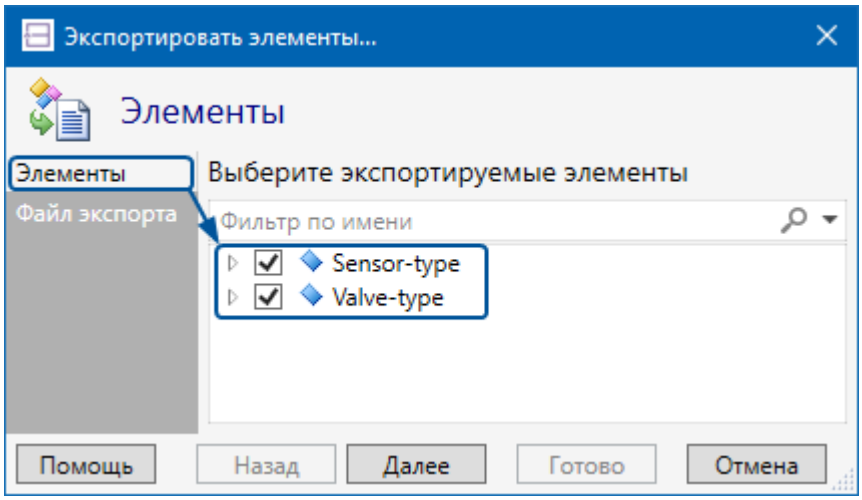



## Экспорт элементов

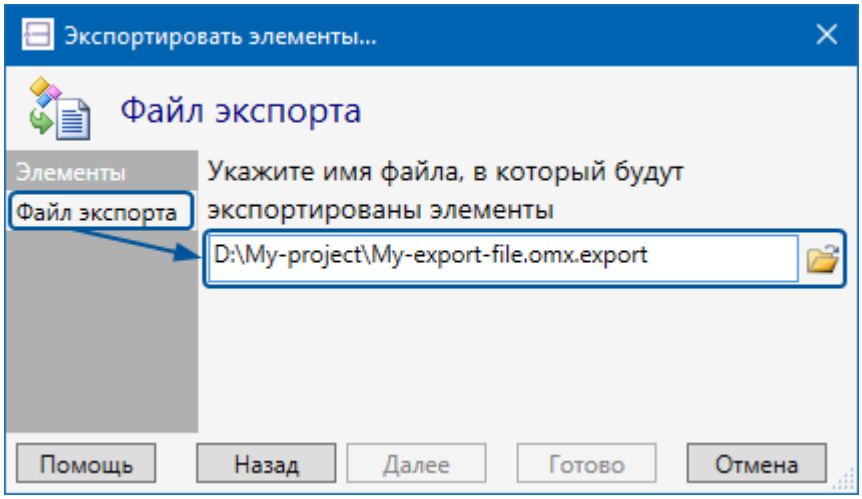
1. В обозревателе решений выбираем пространство имён **«Types»** и в контекстном меню выбираем **Экспортировать элементы...**.



2. В открывшемся окне выбираем «**Sensor-type**» и «**Valve-type**».




3. Указываем файл.




4. Нажимаем **Готово**.

Выбранные элементы будут экспортированы.

## Внесение изменений в файл

В файле один тип удалим, один изменим и ещё один тип добавим, чтобы посмотреть, как каждое изменение повлияет на проект при импорте:

1. Открываем файл в любом текстовом редакторе.

```
<omx xmlns="system" xmlns:ct="automation.control">
  <ct:type name="Sensor-type" uuid="6a5ba1bd-66b8-44dd-9f2e-d43808d938fd">
    <ct:parameter name="Value" access-level="public" access-scope="global"
direction="out" type="float32" uuid="9a2bb99c-ff2d-4a4a-9ee2-b1d42e770b93" />
  </ct:type>
  <ct:type name="Valve-type" uuid="bdecffc4-4874-49a9-b5d4-ba153c665464">
    <ct:parameter name="Is_opened" access-level="public" access-
scope="global" direction="out" type="bool" uuid="7b1ffe2d-4948-4825-a5a5-
7b84f80dc51c" />
    <ct:event name="Open" access-level="public" access-scope="global"
direction="in" type="bool" uuid="a617e3c0-0ccd-48dd-8141-c75b78815025" />
```

```
  </ct:type>
</omx>
```

2. Типу **«Sensor-type»** добавим атрибут удаления: **«isDeleted="true"»**.

```
<omx xmlns="system" xmlns:ct="automation.control">
  <ct:type name="Sensor-type" uuid="6a5ba1bd-66b8-44dd-9f2e-d43808d938fd"
isDeleted="true">
    <ct:parameter name="Value" access-level="public" access-scope="global"
direction="out" type="float32" uuid="9a2bb99c-ff2d-4a4a-9ee2-b1d42e770b93" />
  </ct:type>
  <ct:type name="Valve-type" uuid="bdecffc4-4874-49a9-b5d4-ba153c665464">
    <ct:parameter name="Is_opened" access-level="public" access-
scope="global" direction="out" type="bool" uuid="7b1ffe2d-4948-4825-a5a5-
7b84f80dc51c" />
    <ct:event name="Open" access-level="public" access-scope="global"
direction="in" type="bool" uuid="a617e3c0-0ccd-48dd-8141-c75b78815025" />
  </ct:type>
</omx>
```

3. Типу **«Valve-type»** добавим входящий сигнал **«Close»** типа **«bool»**. Тип сигнала такой же, как у **«Open» - Событие**.

Идентификатор элемента (**uuid**) сигналу можно не указывать: он будет сгенерирован при импорте.

```
<omx xmlns="system" xmlns:ct="automation.control">
  <ct:type name="Sensor-type" uuid="6a5ba1bd-66b8-44dd-9f2e-d43808d938fd"
isDeleted="true">
    <ct:parameter name="Value" access-level="public" access-scope="global"
direction="out" type="float32" uuid="9a2bb99c-ff2d-4a4a-9ee2-b1d42e770b93" />
  </ct:type>
  <ct:type name="Valve-type" uuid="bdecffc4-4874-49a9-b5d4-ba153c665464">
    <ct:parameter name="Is_opened" access-level="public" access-
scope="global" direction="out" type="bool" uuid="7b1ffe2d-4948-4825-a5a5-
7b84f80dc51c" />
    <ct:event name="Open" access-level="public" access-scope="global"
direction="in" type="bool" uuid="a617e3c0-0ccd-48dd-8141-c75b78815025" />
    <ct:event name="Close" access-level="public" access-scope="global"
direction="in" type="bool" />
  </ct:type>
</omx>
```

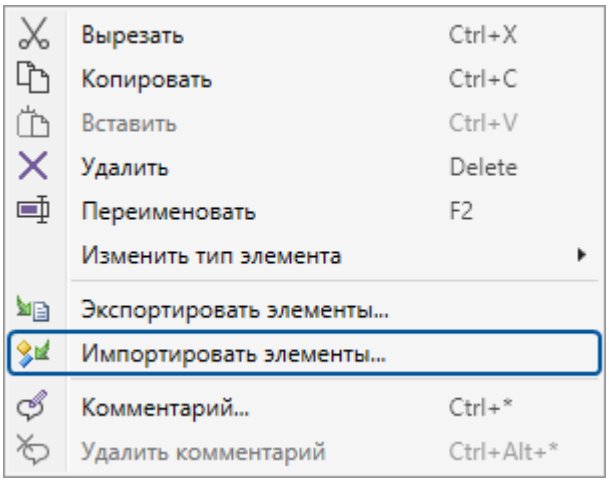
4. Добавим тип **«Device-type»**.

```
<omx xmlns="system" xmlns:ct="automation.control">
  <ct:type name="Sensor-type" uuid="6a5ba1bd-66b8-44dd-9f2e-d43808d938fd"
isDeleted="true">
    <ct:parameter name="Value" access-level="public" access-scope="global"
direction="out" type="float32" uuid="9a2bb99c-ff2d-4a4a-9ee2-b1d42e770b93" />
  </ct:type>
  <ct:type name="Valve-type" uuid="bdecffc4-4874-49a9-b5d4-ba153c665464">
    <ct:parameter name="Is_opened" access-level="public" access-
scope="global" direction="out" type="bool" uuid="7b1ffe2d-4948-4825-a5a5-
7b84f80dc51c" />
    <ct:event name="Open" access-level="public" access-scope="global"
direction="in" type="bool" uuid="a617e3c0-0ccd-48dd-8141-c75b78815025" />
    <ct:event name="Close" access-level="public" access-scope="global"
direction="in" type="bool" />
  </ct:type>
  <ct:type name="Device-type" />
</omx>
```

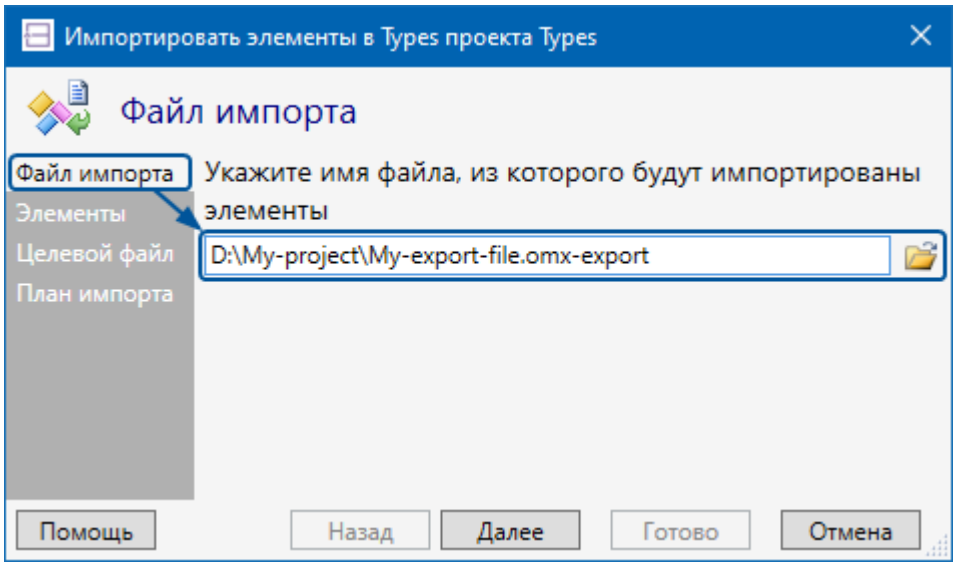
5. Сохраняем изменения в файле.

## Импорт элементов

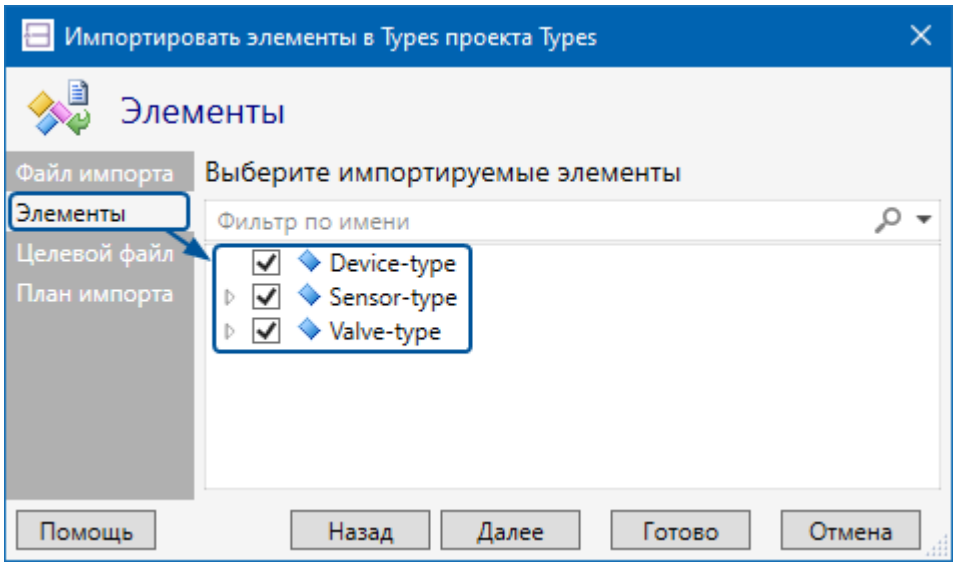
1. В обозревателе решений выбираем пространство имён **«Types»** и в контекстном меню выбираем **Импортировать элементы...**.



2. В открывшемся окне указываем файл.




3. Выбираем элементы в файле, которые будем импортировать.




4. Выбираем файл проекта.

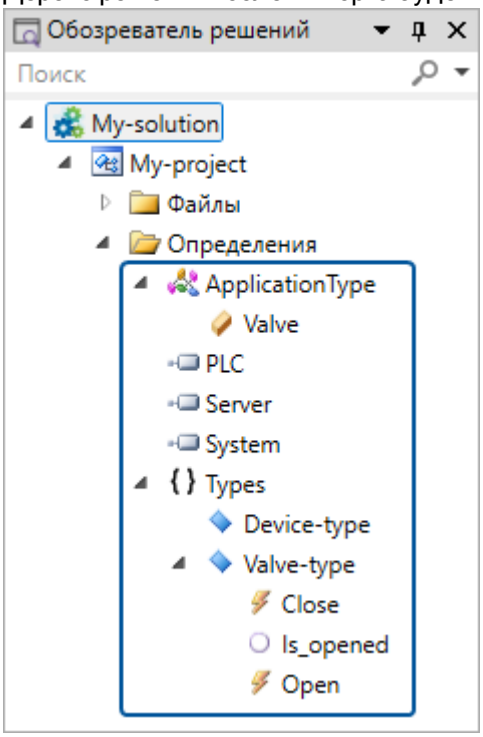
В плане импорта видим изменения, которые произойдут в проекте при импорте.

5. Нажимаем **Готово**.

В результате:

- Тип **«Sensor-type»** будет удалён из проекта. Вместе с ним будет удалён объект этого типа - **«Sensor»**.
- В тип **«Valve-type»** будет добавлен сигнал **«Close»**.
- Будет добавлен тип **«Device-type»**.

Дерево решений после импорта будет иметь вид как на рисунке ниже.



# 6. Развёртывание

В этой главе рассказывается, как запустить решение в среде исполнения.

Чтобы запустить решение, необходимо выполнить:

- построение решения;
- развёртывание решения.

После запуска решения можно выполнить его отладку - проверку корректной работы.

## 6.1. Построение решения

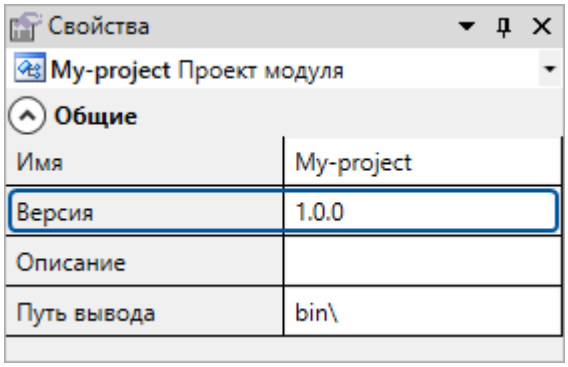
При построении решения будут созданы файлы конфигураций для следующих программных компонентов Систэм Платформ, описанных в решении:

- серверов ввода/вывода SePlatform.Data Server;
- точек доступа SePlatform.AccessPoint.

Для прочих программных компонентов, описанных в решении, конфигурации не создаются, однако их описание используется при создании конфигураций.

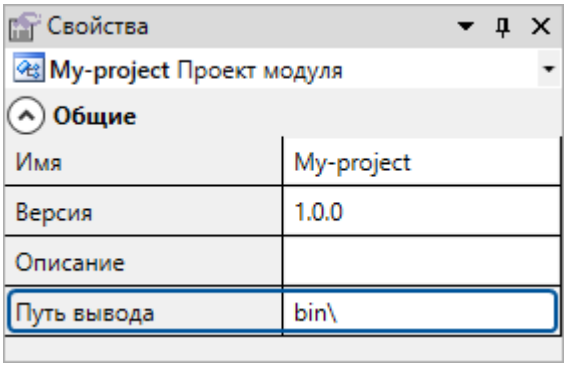
Чтобы построить решение:

1. В свойствах проекта укажите новую версию проекта.



2. В панели инструментов выберите действие **Построить решение**.


Файлы конфигураций будут созданы в папке, указанной в свойствах проекта.




ПРИМЕЧАНИЕ

Перед созданием файлов конфигураций SePlatform.Development Studio выполняет компиляцию решения. Если при компиляции будут обнаружены ошибки, то в журнале сообщений появятся сообщения об ошибках и построение решения прервётся.

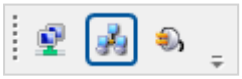
# 6.2. Развёртывание решения

При развёртывании решения к программным компонентам Систэм Платформ, установленным в среде исполнения, будут применены построенные для них конфигурации.

Чтобы развернуть решение:

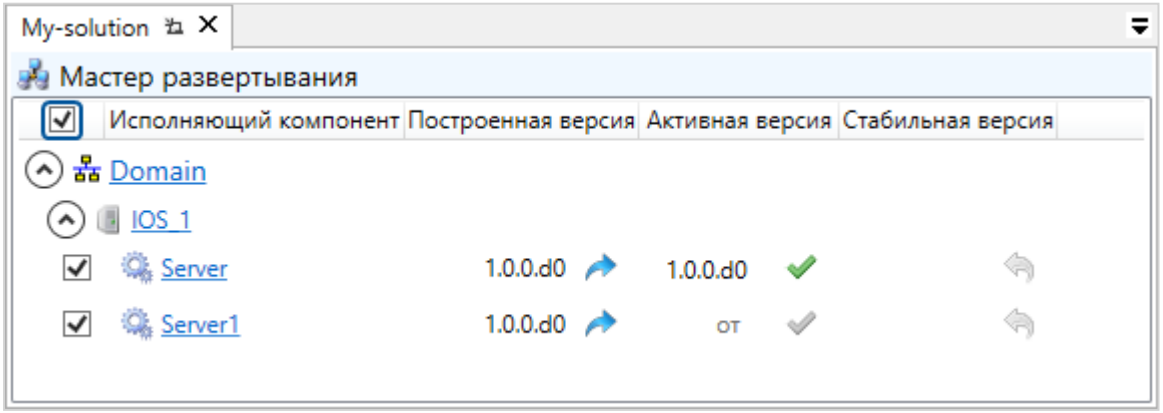
1. Выберите среду исполнения (выполняется на панели инструментов).


2. Запустите **Мастер развёртывания** (выполняется на панели инструментов).



В мастере развёртывания отображается список программных компонентов Систэм Платформ, описанных в решении.

3. Выберите все компоненты, к которым вы хотите применить построенные для них конфигурации.



4. На панели инструментов нажмите кнопку **Применить все**.

После выполнения развёртывания, выбранные компоненты будут сконфигурированы.

# 7. Отладка развёрнутого решения

При отладке можно вручную проверить работу развёрнутого решения:

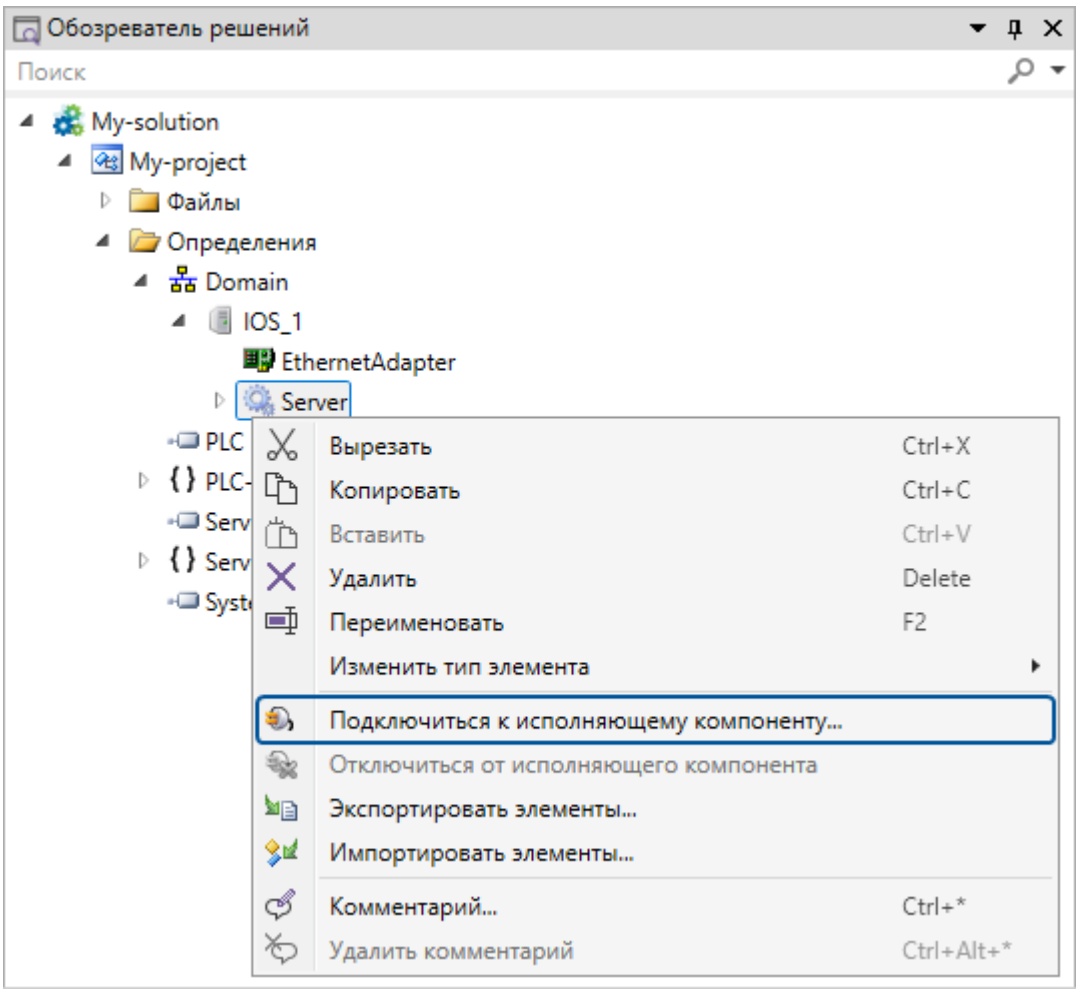
- передачу данных между исполняющими компонентами;
- правильную обработку данных при изменении значений;
- генерацию событий.

Чтобы выполнить отладку, необходимо подключиться к исполняющим компонентам, работающим в среде исполнения, на которых запущено решение. После подключения к исполняющему компоненту в интерфейсе будут доступны:

- просмотр значений параметров/переменных/событий1 у объектов, размещённых на исполняющем компоненте;
- изменение их значений
- возникающие события.

## 7.1. Подключение к исполняющим компонентам

Чтобы подключиться к исполняющим компонентам, запустите **мастер подключения к исполняющим компонентам** из панели управления или из контекстного меню исполняющего компонента и следуйте его инструкциям.




ОБРАТИТЕ ВНИМАНИЕ

Для успешного подключения исполняющий компонент должен иметь один из логических адаптеров: **OPC UA Сервер**, **OPC DA Сервер** или **OPC AE Сервер**.

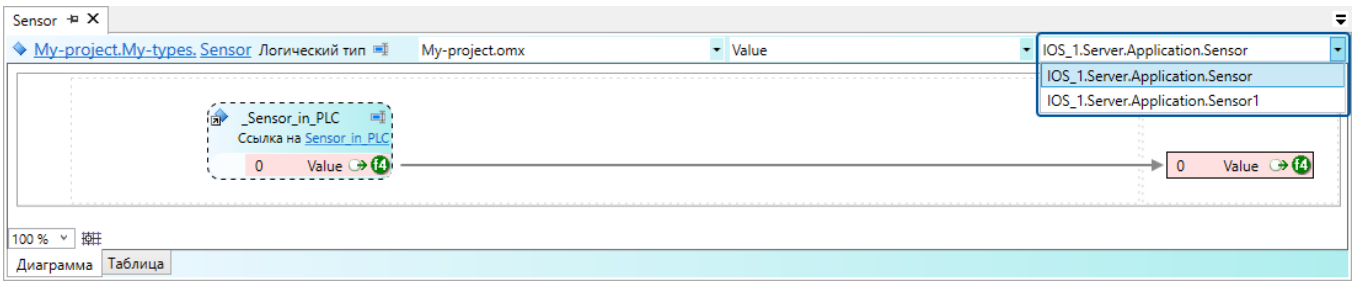
# 7.2. Просмотр и изменение значений

ОБРАТИТЕ ВНИМАНИЕ

Просмотр и изменение значений в исполняющих компонентах выполняются по протоколам OPC UA и OPC DA. Чтобы просматривать/изменять значения в исполняющем компоненте, добавьте в него логический адаптер **OPC UA Сервер** или **OPC DA Сервер**.

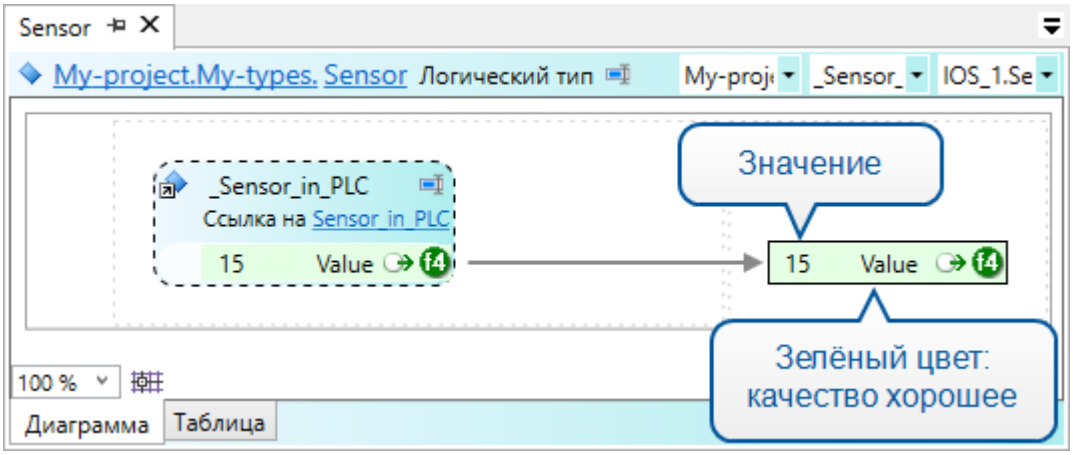
## Просмотр значений

Чтобы посмотреть значения в объекте, размещённом в среде исполнения, перейдите в этот объект или в его тип. Если в среде исполнения размещено несколько таких объектов (на одном исполняющем компоненте или на разных), то выберите конкретный объект в выпадающем списке в правом верхнем углу рабочей области.

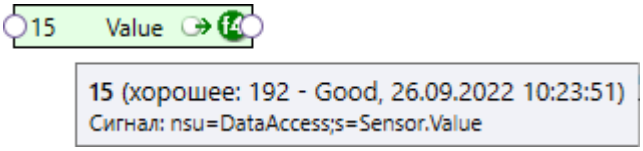



Значения отображаются на элементах параметров, переменных и событий:

- значение отображается возле имени элемента;
- качество значения отображается цветом элемента:
  - зелёный - значение имеет хорошее качество (192) или задано пользователем (216);
  - красный - значение недостоверно.

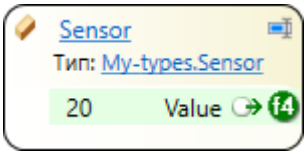



Чтобы посмотреть VQT значения, наведите мышь на значение.



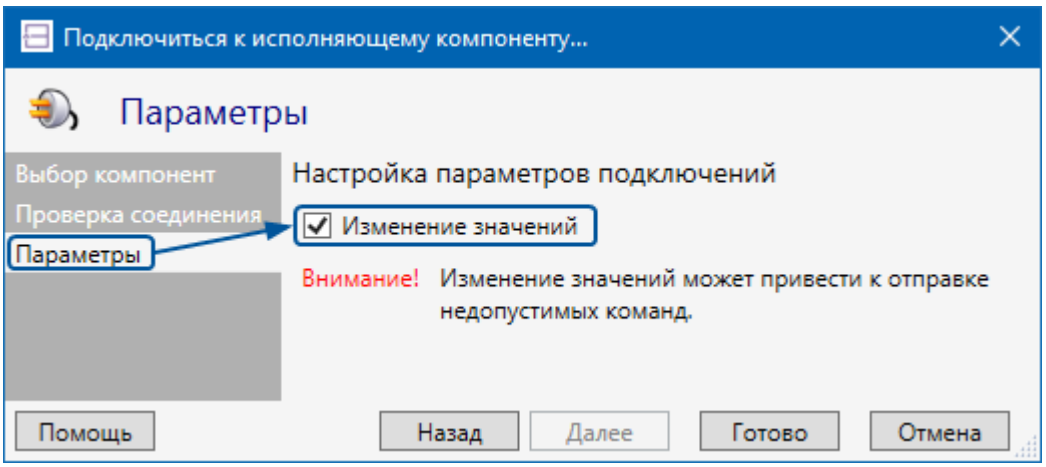

ПРИМЕЧАНИЕ

Просмотр и изменение значений можно осуществлять не только в объекте, но и на изображении этого объекта.




## Изменение значений

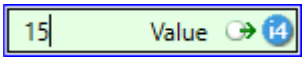
Чтобы изменять значения на исполняющем компоненте, при подключении к нему установите флаг **Изменение значений** на вкладке **Параметры**.



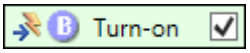

ВАЖНО

Устанавливайте этот флаг только при подключении к тестовой среде исполнения, в которой выполняется проверка работы запущенного решения.

Чтобы изменить значение, введите новое значение в поле значения и нажмите клавишу **«Enter»**.




Чтобы изменить значение булевского типа, установите/снимите флаг значения на элементе.




При изменении значения в исполняющем компоненте будут выполнены все операции, связанные с изменением значения:

- генерация событий;
- выполнение вычислений;
- изменение связанных событий, переменных и параметров.

Все изменения (сгенерированные события и новые значения) отобразятся в SePlatform.Development Studio.

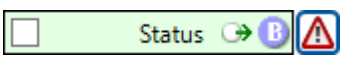
# 7.3. Просмотр событий

ОБРАТИТЕ ВНИМАНИЕ

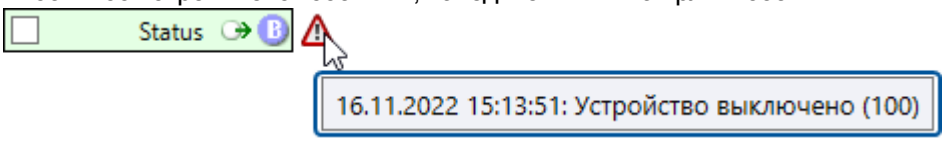
Просмотр событий в исполняющих компонентах выполняется по протоколам OPC UA и OPC AE. Чтобы просматривать события, возникающие в исполняющем компоненте, добавьте в него логический адаптер **OPC UA Сервер** или **OPC AE Сервер**. Для серверов ввода/вывода SePlatform.Data Server и точек доступа SePlatform.AccessPoint наличие логического адаптера **OPC AE Сервер** обязательно: без него события не будут генерироваться в этих исполняющих компонентах.

Чтобы вызвать возникновение события, задайте элементу, для которого настроена генерация событий, значение, которое вызовет генерацию соответствующего события.

Происходящие события отображаются в виде ярлыка возле элемента-источника события.



Чтобы посмотреть текст события, наведите мышь на ярлык события.




ПРИМЕЧАНИЕ

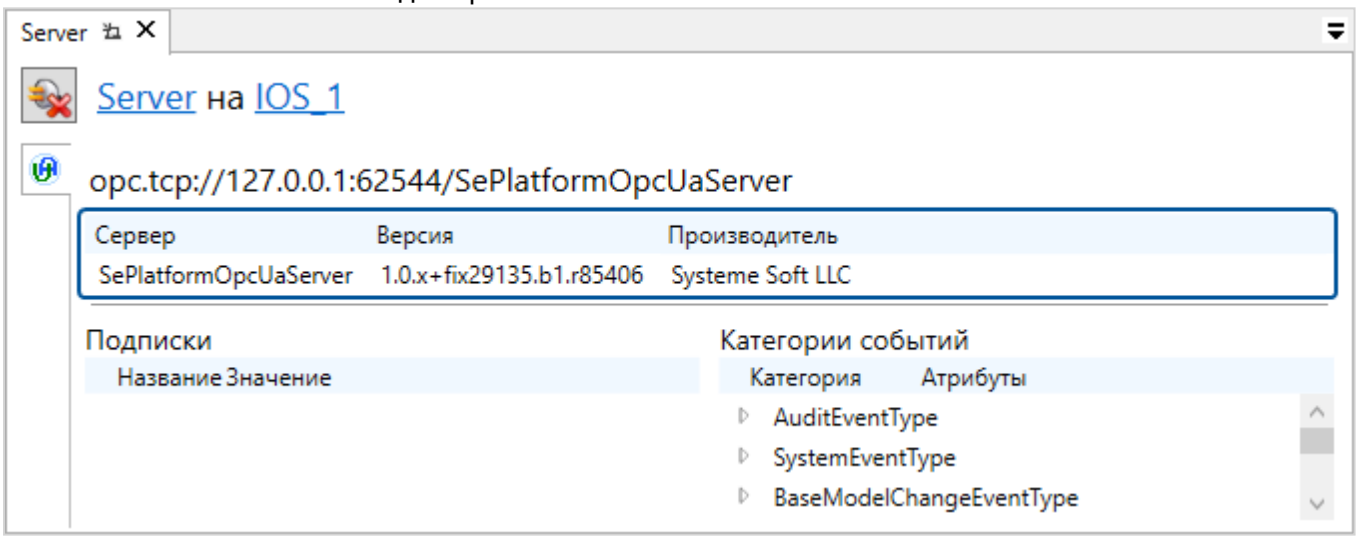
Для каждого элемента отображается только последнее произошедшее событие.

# 7.4. Диагностика подключения

Чтобы проверить состояние подключения к исполняющему компоненту, нажмите по его иконке в строке состояния: отобразится окно состояния подключения. Иконка появляется при подключении к исполняющему компоненту.

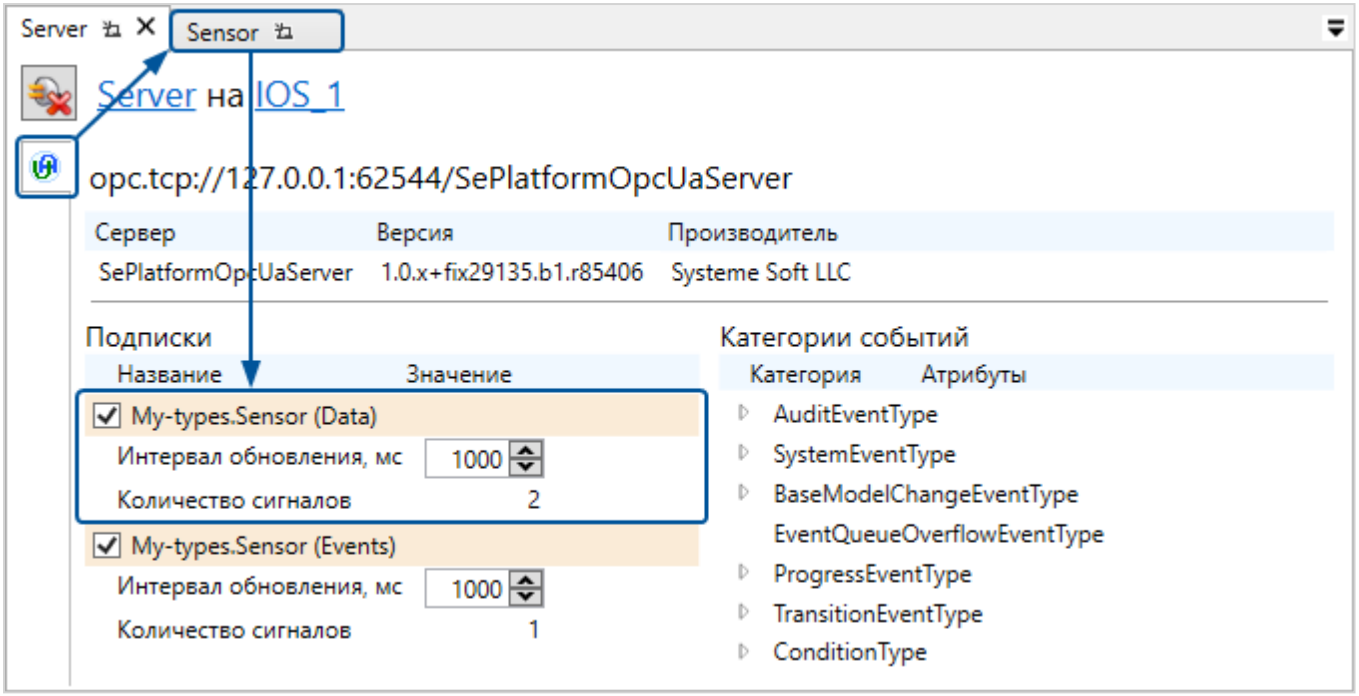
В окне состояния подключения на отдельных вкладках отображается состояние связи с каждым из логических адаптеров **OPC UA Сервер**, **OPC DA Сервер** и **OPC AE Сервер** и параметры получения событий/значений от исполняющего компонента через этот адаптер. Параметры настраиваются отдельно для каждого открытого элемента.

Состояние связи с логическим адаптером:



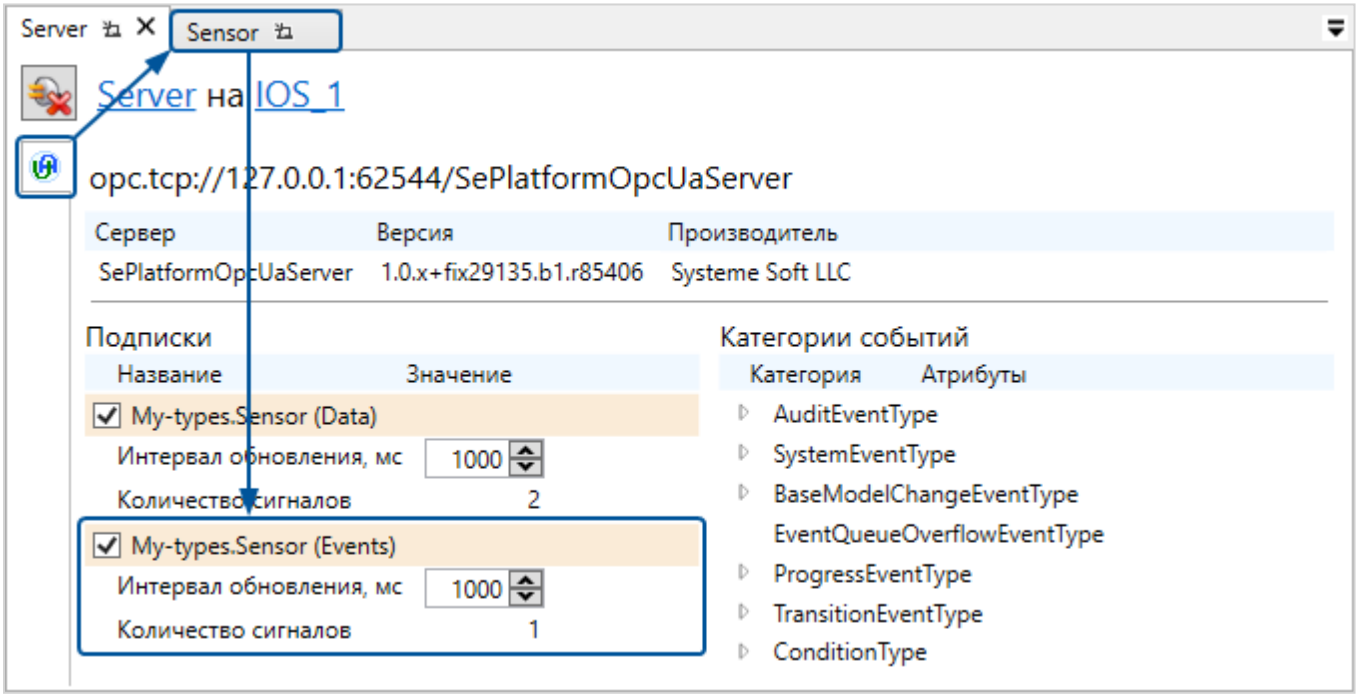

Параметры значений:

- Флаг - получение значений от исполняющего компонента. При снятом флаге просмотр/изменение значений недоступны.
- **Интервал обновления, мс** - интервал между запросами значений от исполняющего компонента.
- **Количество сигналов** - количество элементов, для которых выполняется запрос значений: событий, параметров, переменных.




Параметры событий:

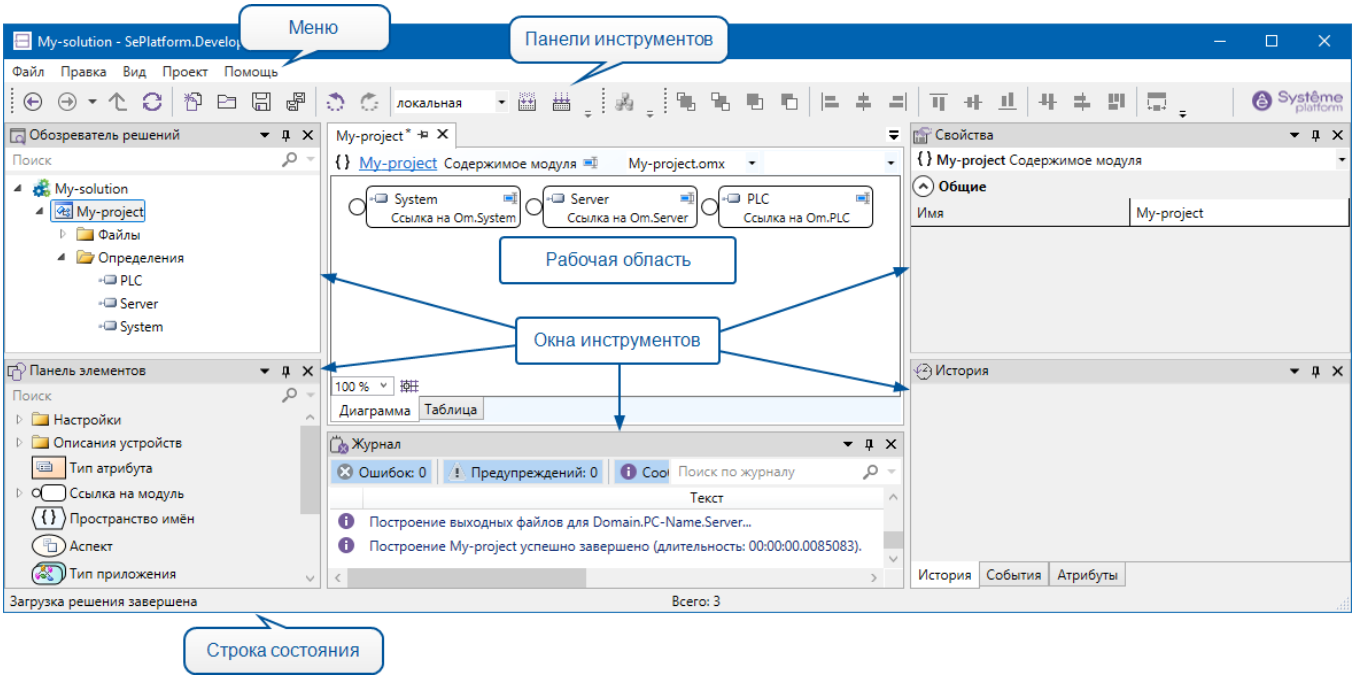
- Флаг - получение событий от исполняющего компонента. При снятом флаге происходящие события не будут отображаться в SePlatform.Development Studio;
- **Интервал обновления, мс** - интервал между запросами событий от исполняющего компонента;
- **Количество сигналов** - количество элементов в объекте, для которых настроена генерация событий.



# 8. Пользовательский интерфейс

Окно SePlatform.Development Studio содержит следующие области:

- Меню
- Панели инструментов
- Рабочая область
- Окна инструментов
- Строка состояния




## Меню

Содержит:

- **Файл** - действия с решением, проектами, открытие параметров приложения
- **Правка** - отмена последних действий, копирование/вставка/удаление элементов, упорядочивание и выравнивание элементов
- **Вид** - включение окон инструментов и панелей инструментов, переход вверх/назад/вперёд по иерархии элементов, сброс макета окон
- **Проект** - компиляция, построение, развёртывание, подключение к исполняющим компонентам, поиск элементов
- **Справка** - открытие встроенной справки, информации о программе

## Панели инструментов

Панели инструментов - группы кнопок быстрых действий:

- Стандартная
- Мастеры

- Диаграмма
- Контекстно-зависимые кнопки - состав группы зависит от элемента, открытого в активной вкладке.

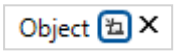
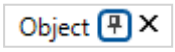
Панели инструментов можно размещать на нескольких строках. Панели инструментов Стандартная, Мастеры и Диаграмма можно включать/отключать в меню **Вид** → **Панели инструментов**.

## Рабочая область

В рабочей области в виде вкладок открываются редакторы.

Вкладки можно разместить в отдельных областях внутри рабочей области или вынести за пределы окна приложения в виде отдельного окна.

Вкладки можно закреплять. Чтобы закрепить вкладку, нажмите иконку на изображении вкладки.

| Состояние | Иконка | Описание |
|---|---|---|
| Режим просмотра |  | Предназначен для того, чтобы не открывать новые вкладки при просмотре элементов без их редактирования. При открытии элемента:<br>- если есть вкладка редактора, открытая в режиме просмотра, элемент откроется в ней;<br>- если вкладки нет - элемент откроется в новой вкладке редактора, открытой в режиме просмотра.<br>Для каждого редактора может быть открыта только одна вкладка в режиме просмотра.<br>Вкладку нельзя переключить в состояние Режим просмотра: вкладка отрывается в этом состоянии по умолчанию и может быть только выведена из него. |
| Закреплена |  | Вкладка остаётся открытой после перезапуска программы.<br>Если находясь на вкладке открыть новый элемент:<br>- если элемент выбран внутри вкладки - вкладка переключится на новый элемент;<br>- если элемент выбран вне вкладки (например в обозревателе решений) - элемент откроется в другой вкладке;<br>- если редактор элемента отличается от редактора, открытого во вкладке - элемент откроется в другой вкладке. |
| Зафиксирована |  | Зафиксированные вкладки ведут себя как закрепленные и отличаются от них следующим поведением:<br>- вкладка не переключается на другой элемент: если выбрать элемент внутри вкладки, он откроется в другой вкладке;<br>- зафиксированные вкладки автоматически размещаются слева от закреплённых. |

### Окна инструментов

Окна инструментов – окна, в которых открываются инструменты, которые есть в SePlatform.Development Studio. Чтобы открыть окно инструмента или переключиться на него, выберите инструмент в меню **Вид**.

Окна инструментов можно разместить в отдельных областях окна приложения или вынести за пределы окна приложения в виде отдельных окон, а также объединить несколько окон инструментов в одно: окна превратятся во вкладки внутри общего окна.

### Строка состояния

В строке состояния отображаются:

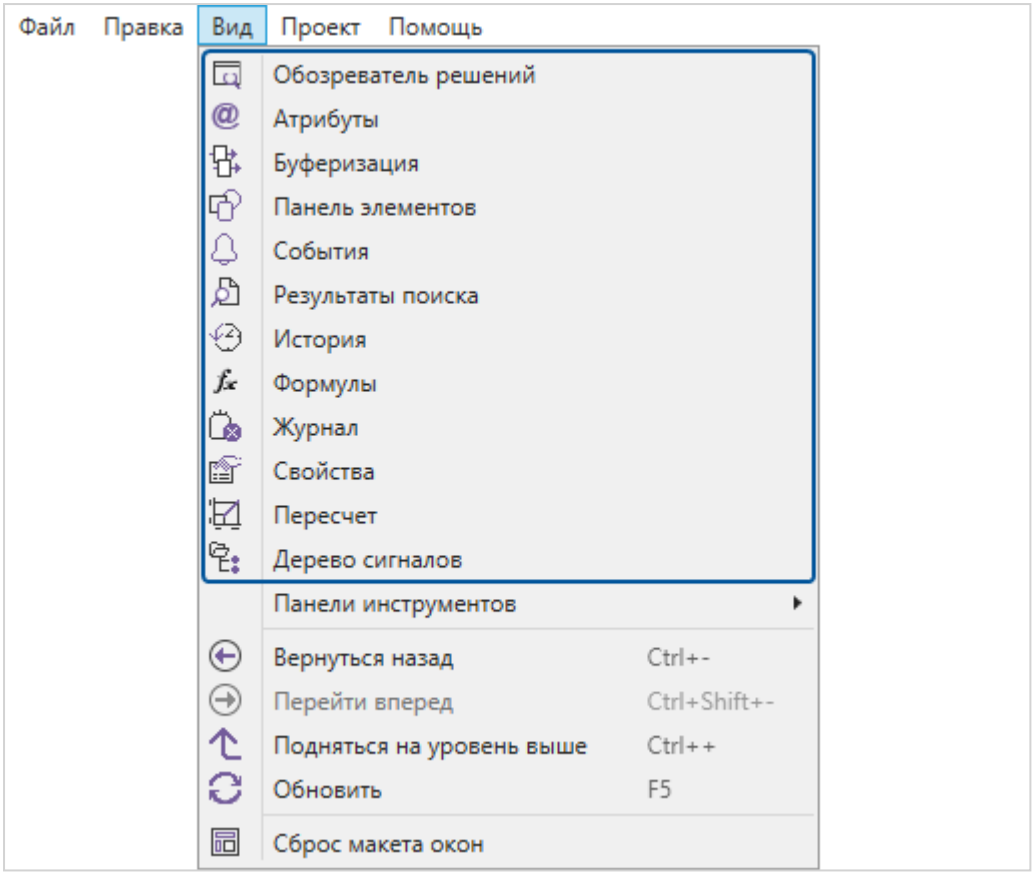
- Иконки связи с исполняющими компонентами

# 8.1. Инструменты

В SePlatform.Development Studio доступны следующие инструменты:

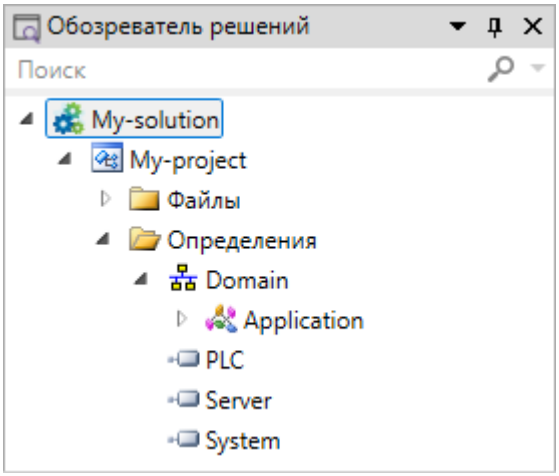
| **Инструмент** | **Назначение** |
|---|---|
| Обозреватель решений | Работа с деревом решения |
| Атрибуты | Работа с атрибутами |
| Панель элементов | Добавление элементов |
| События | Настройка генерации/агрегации событий |
| Результаты поиска | Отображение результатов поиска |
| История | Настройка сохранения значений в историю |
| Формулы | Добавление вычислений |
| Журнал | Отображение сообщений, предупреждений и ошибок, возникающих при работе |
| Свойства | Работа со свойствами элемента |
| Пересчёт | Настройка пересчёта физических единицы в инженерные |
| Дерево сигналов | Имитация дерева сигналов |

Каждый инструмент отображается в собственном окне. Чтобы открыть окно инструмента или переключиться на него, в меню **Вид** выберите инструмент.




## 8.1.1. Обозреватель решений

Окно предназначено для просмотра и редактирования структуры исходных файлов и определений проекта. Структура проекта имеет вид дерева и содержит несколько уровней вложенности объектов.




**Обозреватель решений** допускает одновременное открытие нескольких проектов в рамках одного решения.

В верхней части окна расположена строка поиска, позволяющая фильтровать содержимое дерева проекта в соответствии с искомым текстом.

## 8.1.2. Атрибуты

Окно предназначено для добавления и редактирования атрибутов. Атрибут - это значение, которое указывается элементу и каким-то образом на него влияет:

- одни атрибуты добавляют или изменяют характеристики элемента. Например, атрибут **Начальное значение** задаёт значение сигнала при запуске/перезапустке SePlatform.Data Server.
- другие атрибуты добавляют элементу новое поведение. Например, при наличии у сигнала атрибута **Ведение истории** его значения будут сохраняться в историю.
- есть атрибуты, которые влияют на поведение элемента в проекте. Например, атрибут **Комментарий** добавляет элементу комментарий, который отображается при наведении мышью на элемент.

Все атрибуты можно добавлять и редактировать в окне **Атрибуты**. Также некоторые атрибуты добавляются элементу автоматически при выполнении действий в интерфейсе и работе в других окнах. Например:

- Если в контекстном меню элемента добавить ему комментарий, элементу будет добавлен атрибут **Комментарий**, в котором будет сохранён текст комментария.
- Если в окне **История** сигналу настроить сохранение истории, ему будет добавлен атрибут **Ведение истории**, в значении которого будут заданы параметры, указанные при настройке.

Чтобы использовать атрибуты, нужно добавить в проект ссылки на модули, в которых они находятся: **Om.System** и **Om.Server**. Ссылки на эти модули по умолчанию есть в проектах, созданных из шаблонов: **Библиотека элементов**, **Проект приложения** и **Проект развёртывания**. Чтобы добавить ссылку на модуль:

1. Перейдите в **Содержимое модуля** (корень проекта).
2. Добавьте элемент **Ссылка на модуль**.

   В свойстве **Модуль** укажите нужный модуль.

После добавления ссылки на модуль, в проекте будут доступны атрибуты, которые находятся в этом модуле.

Чтобы добавить элементу атрибут:

1. Выберите элемент.
2. В окне **Атрибуты** в контекстном меню выберите **Добавить новые атрибуты...** или нажмите клавишу **«Insert»**.

   Откроется окно выбора атрибутов, в нём будут только те атрибуты, которые применимы к выбранному элементу.
3. Выберите один или несколько атрибутов и нажмите **Ок**.

   Атрибуты будут добавлены элементу.
4. Каждому атрибуту укажите его значение.

   Формат значения зависит от атрибута: в одних атрибутах нужно указать число, в других - строку, в третьих - структуру данных в определённом формате.

ОБРАТИТЕ ВНИМАНИЕ

В большинстве случаев у элемента не должно быть двух и более атрибутов одного типа.

ОБРАТИТЕ ВНИМАНИЕ

Атрибуты, добавленные типу, при построении применяются к объектам этого типа.

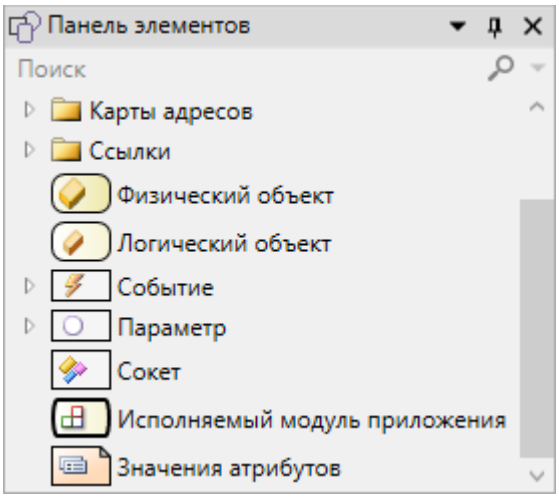
В объекте нельзя редактировать атрибуты вложенных элементов (сигналов, вложенных объектов и других), если они описаны в типе объекта. Есть два способа изменить значения таких атрибутов:

- переопределить в карте значений атрибутов (стр. 100).
- использовать вычисление значения атрибута (стр. 101).

## 8.1.3. Панель элементов

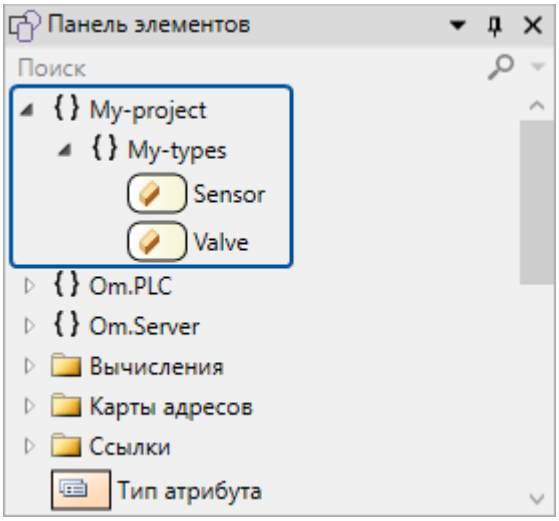
**Панель элементов** предназначена для добавления новых элементов в решение.

На панели элементов отображается список элементов, которые можно добавить в решение. В списке отображаются только те элементы, которые можно добавить в элемент, открытый в рабочей области в настоящее время.

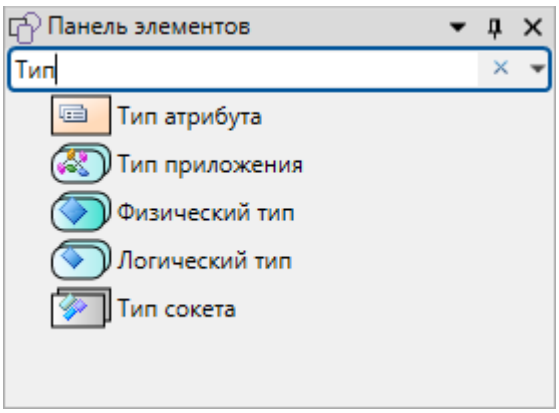



Чтобы добавить элемент в решение, перетащите его из панели элементов в рабочую область.

В списке элементов отображаются объекты добавленных решение типов: это позволяет быстро добавлять в решение объекты нужного типа.

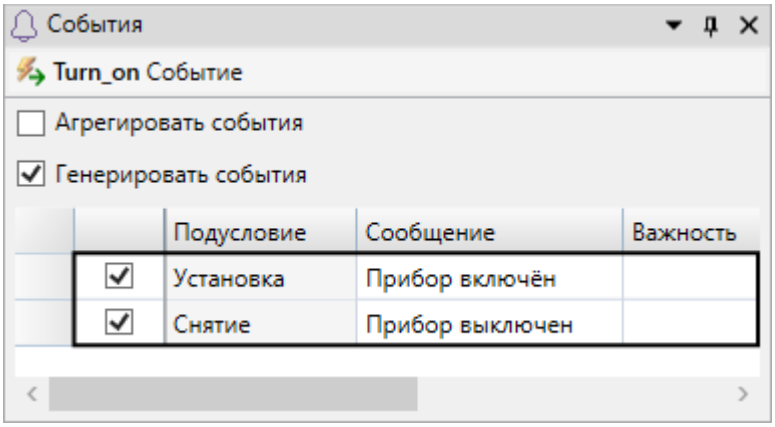



Чтобы найти необходимый элемент, введите его название целиком или частично в поле поиска и нажмите **«Enter»**.




## 8.1.4. События

Окно **События** предназначено для настройки генерации (стр. 80) и агрегации (стр. 99) событий.



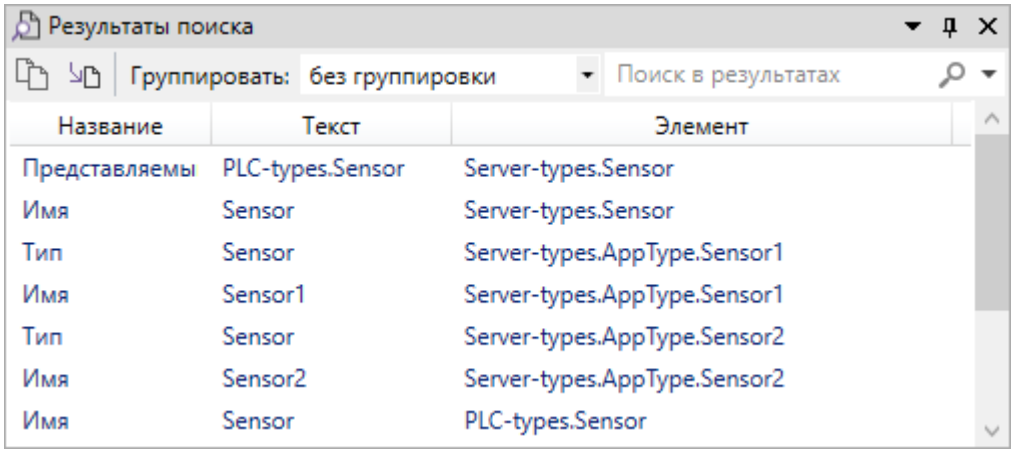

## 8.1.5. Результаты поиска

Чтобы открыть окно или перейти к нему, в меню выберите **Вид** → **Результаты поиска**. Окно автоматически открывается при выполнении поиска или поиска с заменой.

В окне в виде списка отображаются результаты поиска: свойства и атрибуты, в значении которых найдена искомая строка.

ПРИМЕР

Результаты поиска по строке «**Sensor**»:




Параметры элементов списка (колонки в окне):

| Параметр | Описание |
|---|---|
| **Название** | Название свойства или атрибута. |
| **Текст** | Значение свойства или атрибута.<br>Если выполнялся поиск с заменой, в поле будет указано значение до выполнения замены. |
| **Элемент** | Элемент, которому принадлежит свойство или атрибут. |
| **Проект** | Проект, в котором находится элемент. |
| **Файл** | Файл проекта, в котором находится элемент. |

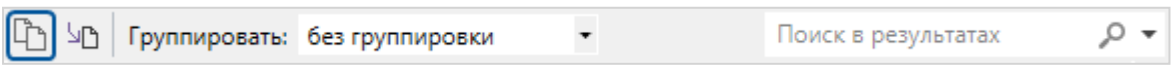
Отображение отдельных параметров можно включить/отключить с помощью контекстного окна в строке заголовков списка.

Доступные действия с элементами списка:

- Перейти к элементу, которому принадлежит свойство или атрибут: дважды кликните по элементу списка.
- Поиск/фильтрация: на панели окна в поле поиска введите строку и нажмите «**Enter**».


В списке останутся элементы списка, в которых будет найдена введённая строка.

- Скопировать: выберите один или несколько элементов списка и нажмите «**Ctrl**» + «**C**» или кнопку на панели окна.




Скопированные элементы списка можно вставить в таблицу Excel.

› Экспортировать в файл: нажмите кнопку на панели окна.


Элементы списка будут сохранены в файл.

ОБРАТИТЕ ВНИМАНИЕ

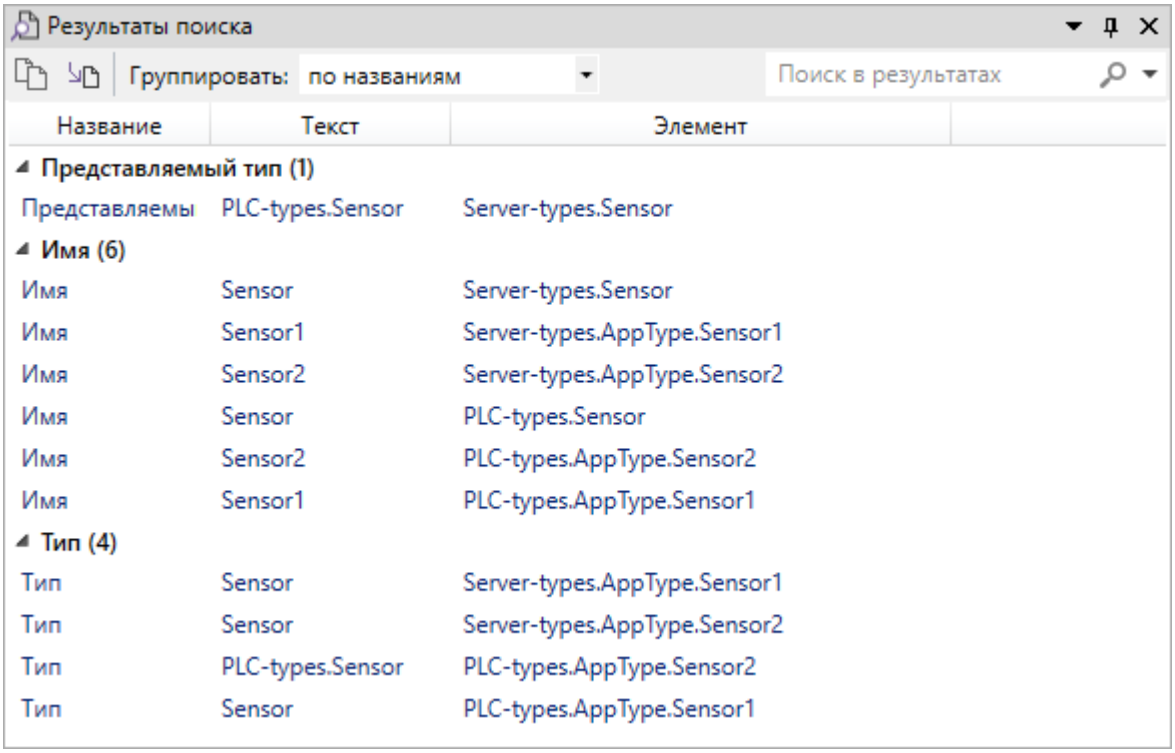
В файл сохраняются все элементы списка, которые отображаются в окне в момент выполнения экспорта. Если выполнен поиск/фильтрация по списку, то в файл попадут результаты поиска/фильтрации.

› Группировать: на панели окна в выпадающем списке выберите параметр, по которому нужно группировать результаты поиска.


Результаты поиска будут разбиты на группы: в каждую группу попадут свойства и атрибуты с одинаковым значением выбранного параметра. Имя группы - значение параметра.

ПРИМЕР

Группировка на названиям:



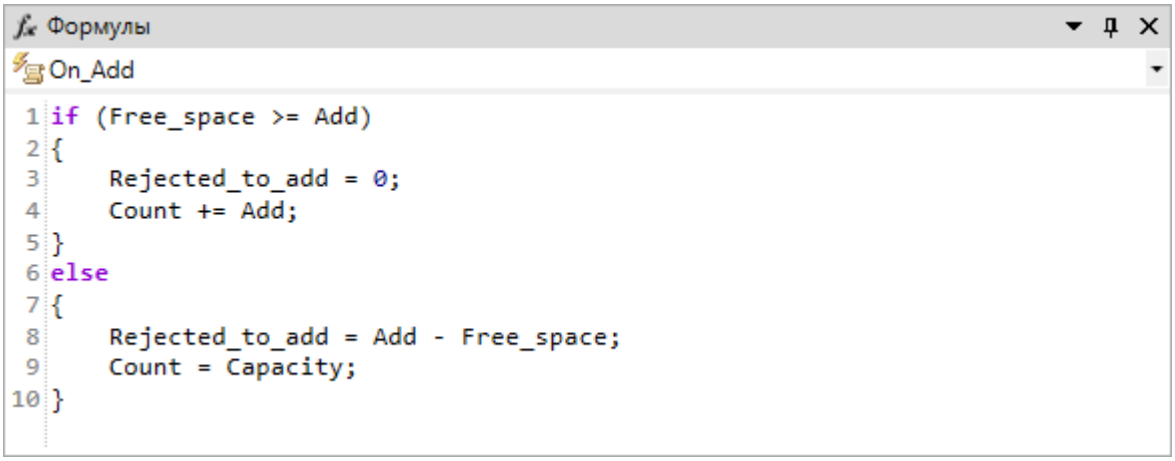

## 8.1.6. История

Окно предназначено для настройки сохранения значений сигналов (стр. 94).

Параметры, указанные в окне **История**, переносятся в атрибут сигнала **Ведение истории** (атрибут добавляется автоматически).

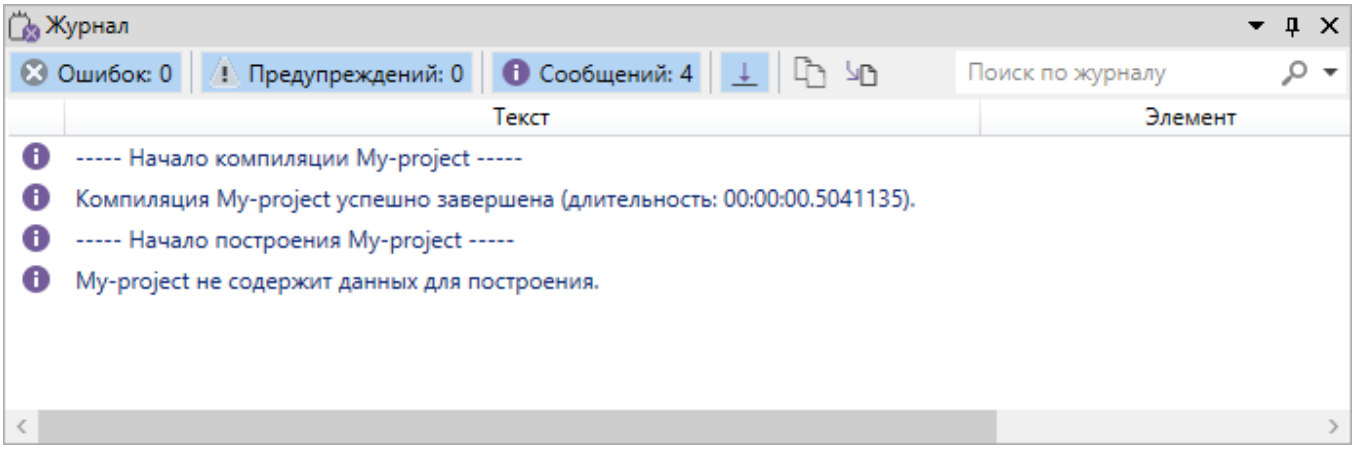
## 8.1.7. Формулы

Окно **Формулы** предназначено для добавления в решение вычислений на языке SePlatform.Om (стр. 74).




## 8.1.8. Журнал

Окно **Журнал** предназначено для отображения информации, отражающей характер течения процесса построения и сборки проекта.




Уведомления могут быть следующих видов:

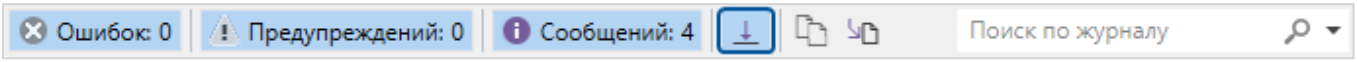
- **Сообщения** - оповещения информационного характера. Свидетельствуют о завершении очередного этапа компиляции проекта;
- **Предупреждения** - сообщения некритического характера, оповещают о возникновении незначительных ошибок при формировании и компиляции проекта, влекущих за собой некорректную работу отдельных его частей;
- **Ошибки** - критические неполадки, возникшие в процессе сборки проекта. Содержат информацию о неверном построении отдельных частей проекта.

Оповещения расположены в виде табличного списка. В столбцах таблицы содержатся следующие сведения:

- **Описание** - столбец, отражающий основной текст уведомлений;
- **Элемент** - столбец, отражающий путь к элементу в рамках проекта;
- **Файл** - столбец, отражающий полный путь к файлу, в котором определен элемент;
- **Проект** - столбец, отражающий наименование проекта.

Кнопки **Ошибок**, **Предупреждений**, **Сообщений** отражают количество каждого вида сообщений. В случае значительного количества сообщений есть возможность включить отображение только одного типа уведомлений. Для этого необходимо нажать соответствующую кнопку. Повторное нажатие кнопки восстанавливает отображение содержимого таблицы полностью.

Кнопка **Автоматическое прокручивание списка** исключает необходимость ручного прокручивания списка сообщений. При нажатии кнопки автоматически выполняется переход к конку списка. При добавлении в журнал новых сообщений последнее сообщение, добавленное в список, всегда остается видимым.

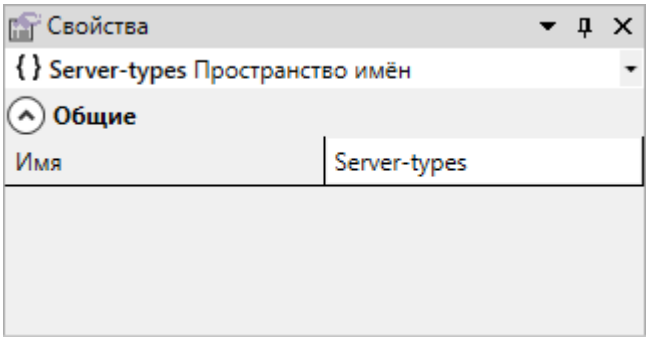



В области журнала действуют следующие комбинации клавиш:

- **«Ctrl» + «A»** - выделить все сообщения в списке
- **«Ctrl» + «C»** - скопировать выделенные сообщения

## 8.1.9. Свойства

Окно **Свойства** предназначено для настройки свойств элементов проекта




Каждый элемент имеет индивидуальные свойства. Для настройки свойств элемента следует выделить нужный объект и заполнить необходимые поля в окне. Элемент с заданными свойствами включается в состав проекта и автоматически связывается с контактирующими элементами. В случае наличия ошибки при задании значения свойства поле соответствующего значения и элемент выделяются красной рамкой.

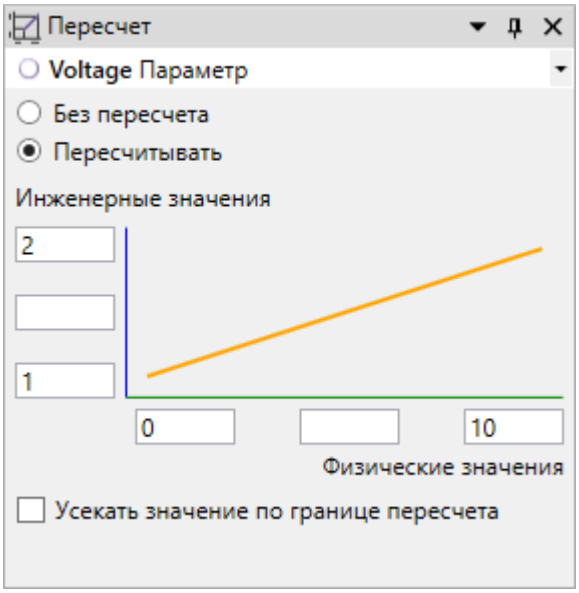
В качестве защиты от ошибок SePlatform.Development Studio позволяет:

- Не указывать наименование элемента. При пустом значении в качестве имени элемента будет отображаться его наименование по умолчанию.
- Выбирать значение свойства из выпадающего списка.
- Выбирать элемент в диалоговом окне - доступно для свойств, ссылающихся на элементы проекта.

В диалоговом окне отображаются только те элементы, которые можно указать в свойстве.

## 8.1.10. Пересчёт

Окно позволяет задавать свойства пересчета физических значений сигналов в инженерные и в обратную сторону.

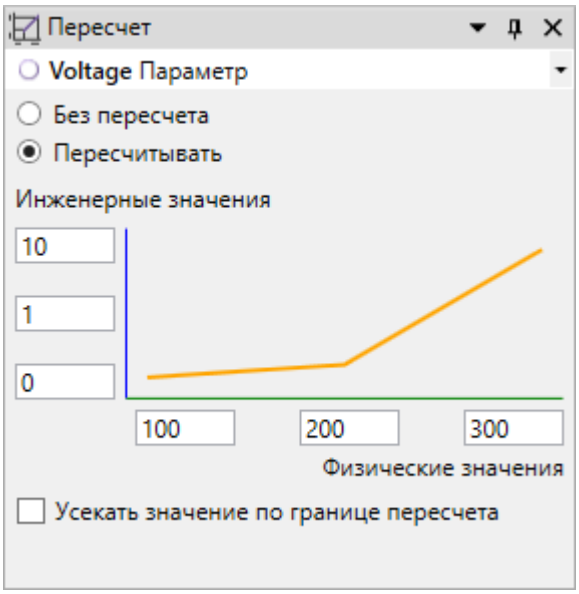



Свойства пересчета можно задать только для элементов типа **Параметр** и **Событие**. В зависимости от типа сигналов возможны следующие варианты:

- **«Без пересчета»** - значения сигнала не пересчитываются;
- **«Инвертировать»** - вариант возможен только для параметров и событий типа **bool**. Для сигналов другого типа возможность выбора данной позиции отсутствует. В результате для свойства сигнала RecalcInvert задается значение **true**;

- **«Пересчитывать»** - для сигнала указываются следующие свойства:
  - нижняя граница физического значения (свойство RecalcRawLow);
  - граница излома физического значения (свойство RecalcRawMiddle);
  - верхняя граница физического значения (свойство RecalcRawHigh);
  - нижняя граница инженерного значения (свойство RecalcValLow);
  - граница излома инженерного значения (свойство RecalcValMiddle);
  - верхняя граница инженерного значения (свойство RecalcValHigh).

Согласно указанным значениям строится график, отражающий зависимость физических значений сигнала от инженерных и наоборот. В зависимости от присутствующих свойств сигнала выполняется линейный пересчет либо линейный пересчет с изломом.



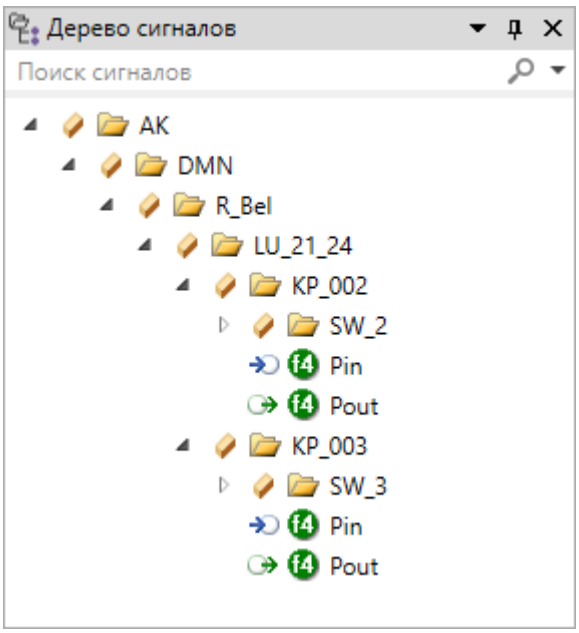

В случае установки флага **Усекать значение по границе пересчета** результирующее значение сигнала при выходе за диапазон пересчета приравнивается к ближайшей границе пересчета. Данный флаг добавляет сигналу свойство RecalcTruncate.

При выходе значения за границы пересчета для сигнала может устанавливаться плохое качество. Данное поведение определяется свойством сигнала RecalcSetFailureQuality. Для добавления сигналу данного свойства следует установить флаг **При усечении выставлять плохое качество**.

Ранги пересчета, автоматически переносятся в качестве значений атрибута Recalc в окне **Атрибуты**.

## 8.1.11. Дерево сигналов

Окно предназначено для имитации дерева сигналов.




Дерево сигналов строится в соответствии со значениями атрибута **Имя узла в OPC** для объектов проекта автоматизации. При построении учитывается вложенность объектов. Тег сигнала формируется на основе наименований узлов высших уровней иерархии.
Содержимое окна меняется в зависимости от выделенного объекта. При выделении узла все его дочерние объекты отображаются в виде вложенных узлов.

# 8.2. Редакторы

## 8.2.1. Редактор элемента

Под редактированием содержимого элемента следует понимать редактирование состава вложенных элементов и настройка их свойств. Отдельный элемент редактируется в собственном редакторе. Редактором называется отдельная закладка с содержимым элемента. Можно открыть одновременно несколько редакторов для разных элементов.

Переход к редактору элемента может выполняться с помощью:

- нажатия на ссылку с наименованием элемента, расположенную в верхней части образа элемента;
- двойного нажатия на элемент в **Обозревателе решений**;
- выбора команды **Перейти к определению** в контекстном меню
- клавишей **«F12»**

В верхней части редактора элемента отображается информация:

- составное имя элемента с учетом его принадлежности к верхним уровням иерархии проекта. Для разделения уровней расположения элемента в дереве проекта используются точки;
- кнопка переименования элемента;
- поле для навигации по файлам проекта приложения;
- поле для навигации по элементам выбранного файла.

Содержимое элемента можно представить в двух видах:

- диаграмма (вид по умолчанию);
- таблица.

Переключение между видами проводится в нижней части области просмотра и редактирования.

## Диаграмма

Диаграмма элемента отражает весь состав вложенных элементов и связи между ними. Каждый отдельный элемент отображается в виде образа элемента.
Если содержимое диаграммы не помещается на экран и появляются полосы прокрутки. В этом случае перемещаться по ней можно:

- С помощью полос прокрутки.
- С помощью горячих клавиш:

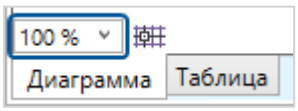
| **Клавиша или комбинация клавиш** | **Результат** |
|---|---|
| Стрелки | Переместиться в направлении стрелки. |
| PageUp | Переместиться на страницу выше. |
| PageDown | Переместиться на страницу ниже. |
| Home | Переместиться до конца влево. |
| End | Переместиться до конца вправо. |
| Ctrl + Home | Переместиться до конца вверх. |
| Ctrl + End | Переместиться до конца вниз. |

- С помощью мыши:

| **Действие** | **Результат** |
|---|---|
| Прокрутка колёсика мыши | Переместиться выше/ниже. |
| Alt + прокрутка колёсика мыши | Переместиться влево/вправо. |
| Ctrl + зажатая левая клавиша мыши | Переместиться, захватив и переместив точку под курсором. |

Изменять масштаб можно:

- С помощью поля задания масштаба внизу диаграммы.




- С помощью прокрутки колёсика мыши с зажатой клавишей Ctrl.

## Таблица

Табличный вид диаграммы элемента отражает весь набор вложенных элементов в виде списка. В отличие от диаграммы, таблица позволяет одновременно увидеть:

- все дочерние элементы, вплоть до самого нижнего уровня иерархии;
- свойства и атрибуты всех дочерних элементов.

В таблице доступны следующие действия:

- редактирование значений. Редактировать можно значения в ячейках, выделенных белым цветом. При редактировании можно использовать буфер обмена;
- фильтрация. Фильтрация доступна в каждом столбце и проводится по значениям в строках. При фильтрации можно пользоваться выпадающим списком либо вводить собственное условие фильтра (в т.ч. и регулярные выражения);
- навигация по элементам. Для перехода к таблице отдельного элемента следует щелкнуть по заголовку строки или нередактируемой ячейке;
- добавление/удаление элементов (действия - аналогично диаграмме). Любой элемент добавляется в верхний уровень иерархии отображаемых элементов.

Контекстное меню элементов в табличном виде полностью совпадает с контекстным меню элементов в виде диаграмм.

## 8.2.2. Редактор исходного кода

Редактор исходного кода - текстовый редактор, предназначенный для работы с исходным кодом, написанном на языке SePlatform.Om.

Редактор открывается в виде вкладки в рабочей области. Чтобы открыть редактор:

- кликните по заголовку обработчика события.
- дважды кликните по обработчику события в обозревателе решений.

Полезные функции редактора:

- подсветка синтаксиса.
- автодополнение - при наборе имени редактор предлагает подходящие ключевые слова и имена сигналов.

## 8.2.3. Редактор карт адресов

Редактор карт адресов предназначен для работы с картами адресов, добавленными в решение. Окно редактора карт адресов открывается в отдельной вкладке в основной рабочей области при открытии карты адресов, добавленной в решение. В окне в табличном виде отображается список сигналов того экземпляра приложения, в котором находится данная карта адресов.

Для поиска сигналов в редакторе используется фильтрация: фильтрация доступна в каждом столбце и проводится по значениям в строках. При фильтрации можно пользоваться выпадающим списком либо вводить собственное условие фильтра (в т.ч. и регулярные выражения).

Столбцы **Сигнал**, **Тип** и **Привязка** описывают сигнал и являются общими для всех карт связывания:

- **Сигнал** - OPC-тег элемента.
- **Тип** - тип сигнала.

- **Привязка** - способ задания адреса элемента в карте адресов:
  - **«не привязан»** - адрес не указан в карте адресов, редактировать параметры адреса нельзя.
  - **«непосредственно»** - адрес указан в данной карте адресов.
  - **«только чтение»** - адрес указан в карте адресов, но изменять его параметры в редакторе нельзя. Указывается, чтобы предотвратить непреднамеренное изменение адреса.
  - **«унаследован»** - адрес задан в наследуемой карте адресов, редактировать параметры адреса нельзя.
  - **«переопределен»** - адрес задан в наследуемой карте адресов и может быть переопределён в данной карте адресов.

Остальные столбцы представляют собой отдельные параметры адреса сигнала. Список параметров адреса зависит от карты адресов.

Для поиска элементов в редакторе используется фильтрация: фильтрация доступна в каждом столбце и проводится по значениям в строках. При фильтрации можно пользоваться выпадающим списком либо вводить собственное условие фильтра (в т.ч. и регулярные выражения).

## 8.2.4. Редактор карт атрибутов

Редактор карт атрибутов предназначен для работы с картами атрибутов, добавленными в решение.

Окно редактора карт атрибутов открывается в отдельной вкладке в основной рабочей области при открытии карты значений атрибутов.

Для поиска элементов в редакторе используется фильтрация: фильтрация доступна в каждом столбце и проводится по значениям в строках. При фильтрации можно пользоваться выпадающим списком либо вводить собственное условие фильтра (в т.ч. и регулярные выражения).

Использование карты атрибутов описано в разделе 5.6. Переопределение значений атрибутов с помощью карты атрибутов (стр. 100).

## 8.2.5. Схема представлений

Схема представлений предназначена для отображения всех представлений объекта/типа и связей между ними.

Чтобы открыть схему представлений, выберите объект/тип и нажмите кнопку **Схема представлений** на панели инструментов.

Схема представлений открывается в отдельной вкладке рабочей области. Название вкладки - имя объекта/типа, для которого открыта схема представлений.

На схеме представлений отображаются:

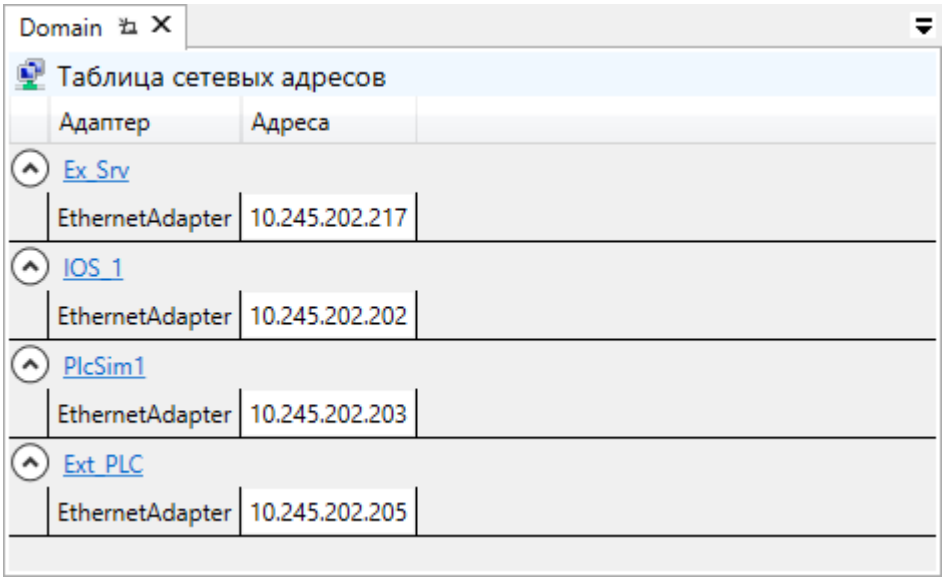
- Аспекты, в которых представлен объект/тип - отображаются в виде вертикальных регионов.
- Представления объекта/типа.

Каждое представление отображается в регионе своего аспекта; представления без аспектов отображаются в левой части в неаспектном регионе.

- Связи - направлены от объекта/типа, содержащего ссылку, к объекту/типу, на который эта ссылка указывает. Цвет связи:
  - голубой - между ссылкой и типом.
  - золотой - между ссылкой и объектом.
  - оранжевый - между ссылкой и объектом, которым она инициализирована.

## 8.2.6. Таблица сетевых адресов

Редактор предназначен для просмотра и редактирования IP-адресов устройств в домене. В редакторе в виде списка отображаются все Ethernet-адаптеры домена.




Чтобы открыть таблицу сетевых адресов, нажмите кнопку **Таблица сетевых адресов** на панели инструментов.

Таблица сетевых адресов содержит следующие столбцы:

- **Адаптер** - наименование сетевого адаптера;
- **Адреса** - сетевой адрес (адреса) устройства.

Каждому адаптеру можно задать один или несколько сетевых адресов:

- IP-адреса разделяются с помощью прямой черты «**|**»
- у источника нескольких каналов передачи данных их адреса следует разделять запятой

Сетевые адреса можно вставить из таблиц Excel. Для этого необходимо скопировать строки со значениями адресов в буфер обмена, выделить одну из ячеек, содержащую адрес в таблице адресов и нажать «**Ctrl**» + «**V**».

## 8.2.7. Мастер развёртывания

Мастер развертывания представляет собой вспомогательный компонент среды разработки, предназначенный для работы с конфигурациями SePlatform.Data Server, сформированными при построении решения.

С помощью мастера развертывания возможно выполнение следующих действий:

- применение разработанной конфигурации;
- подтверждение стабильности текущей конфигурации;
- возврат к последней стабильной версии конфигурации;
- индикация различия построенной и текущей конфигураций.

Для открытия мастера развертывания следует нажать кнопку **Мастер развертывания** на панели инструментов.

Основная область мастера развертывания представляет собой таблицу, в строках которой содержится перечень конфигурируемых исполняющих компонентов, описанных в решении: серверов ввода вывода и точек доступа.

Объединяющая строка, содержащая имя домена, содержит также адрес домена в сети SePlatform.Net. Отображаемый адрес указывается в свойствах элемента **Домен**. Наименования вычислителей и доменов, отображаемые в мастере развертывания, имеют вид ссылок, с помощью которых можно перейти к дигаграммам соответствующих элементов.

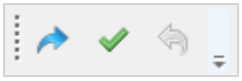
Описание столбцов:

- **Исполняющий компонент** – имя исполняющего компонента в описании домена;
- **Построенная версия** – номер версии разработанной конфигурации. Значение задается в свойствах проекта и обновляется после построения решения;
- **Активная версия** – номер текущей конфигураций, работающей на SePlatform.Data Server. Версию активной конфигурации можно выбрать в выпадающем списке, доступном при нажатии на поле значения;
- **Стабильная версия** – номер конфигурации, отмеченной как стабильная. Стабильной считается конфигурация, не содержащая ошибок.

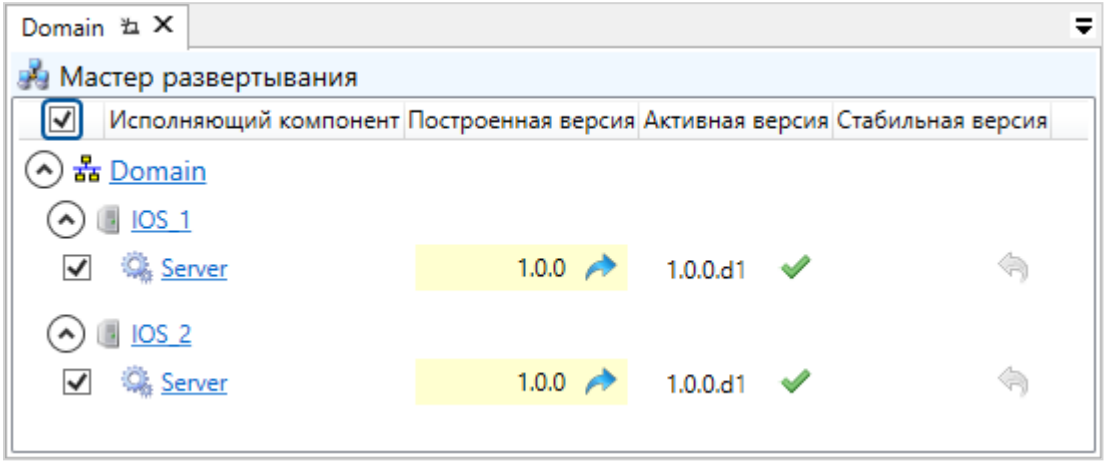
В зависимости от вида конфигурации для каждого вычислителя возможно выполнить следующее:

- использовать построенную конфигурацию в качестве рабочей. Для применения следует нажать кнопку (Применить конфигурацию). В результате нажатия кнопки номер активной конфигурации на вычислителе будет заменен на номер версии, указанной в проекте. Кнопка недоступна, если версия построенной конфигурации и версия активной конфигурации совпадают;
- зафиксировать стабильность активной конфигурации на сервере. Для подтверждения следует нажать кнопку (Подтвердить, что активная конфигурация стабильна). В результате подтверждения версия стабильной конфигурации на сервере будет заменена на номер отмеченной конфигурации. Кнопка недоступна, если конфигурация уже отмечена как стабильная;
- возвратиться к последней стабильной конфигурации. Для возврата следует нажать кнопку (Вернуть стабильную конфигурацию). В результате возврата версия активной конфигурации на сервере будет заменена на последнюю конфигурацию, зафиксированную в качестве стабильной. Кнопка недоступна, если указана верная стабильная конфигурация.

Операции с конфигурациями можно производить также с помощью кнопок, расположенных на панели инструментов.

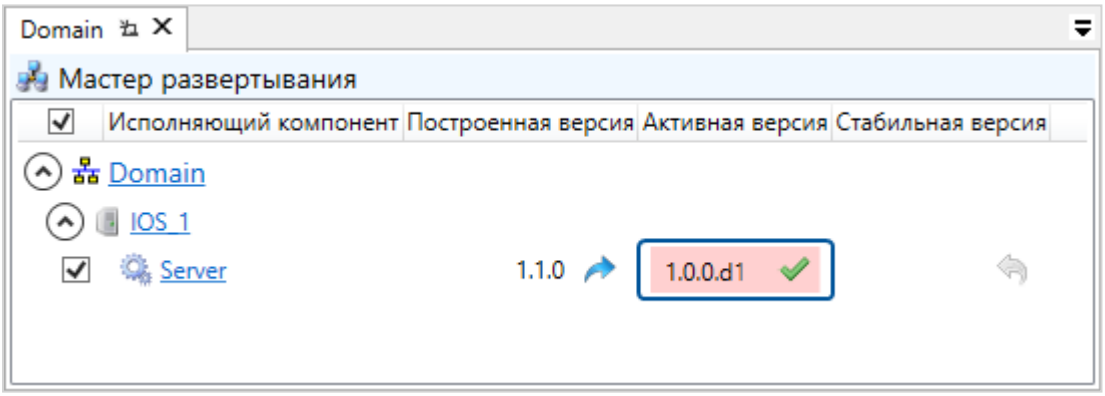


Команды кнопок аналогичны командам кнопок, расположенных непосредственно внутри таблицы мастера развертывания. Основным предназначением кнопок на панели инструментов является возможность одновременного выполнения команды для всех отмеченных вычислителей. Указание выполняется с помощью установки флага рядом с каждым вычислителем. Для того чтобы одновременно отметить все вычислители, следует установить флаг в левом верхнем углу мастера.

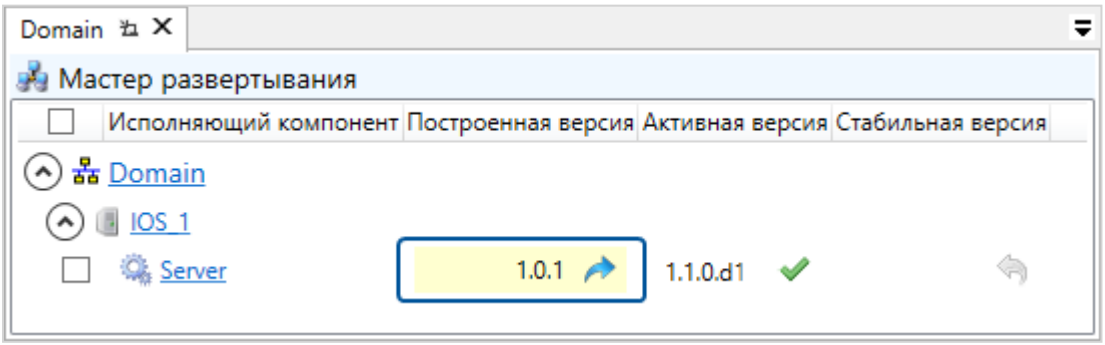



Для наглядного отображения различий рабочей конфигурации и конфигурации, сформированной в проекте, используется следующая цветовая индикация:

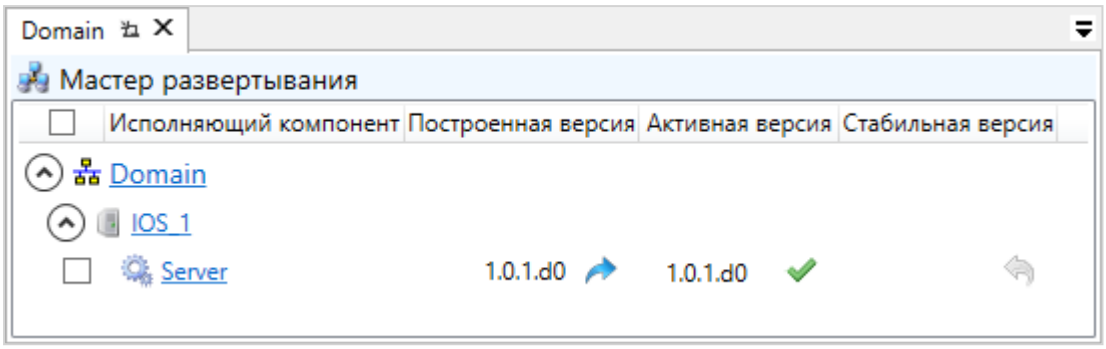
- в случае если рабочая конфигурация сервера младше построенной конфигурации, номер версии активной конфигурации окрашивается в красный цвет




- в случае если рабочая конфигурация сервера старше построенной конфигурации, номер версии построенной конфигурации окрашивается в желтый цвет



› в случае если номера версий рабочей и построенной конфигурации совпадают, цветовая индикация отсутствует




# 8.3. Горячие клавиши

| | |
|---|---|
| **Приложение** | |
| Создать решение | Ctrl + N |
| Открыть решение | Ctr + O |
| Сохранить | Ctrl + S |
| Сохранить все | Ctrl + Shift + S |
| Вызвать контекстную справку | F1 |
| **Решение** | |
| Компилировать решение | F7 |
| Построить решение | F6 |
| Прервать компиляцию/построение | Ctrl + Break |
| **Навигация в проекте** | |
| Вернуться назад | Ctrl + - |
| Перейти вперёд | Ctrl + Shift + - |
| Подняться на уровень выше | Ctrl + + |
| **Навигация в рабочей области** | |
| Горячие клавиши, используемые для навигации в рабочей области, описаны тут (стр. 177). | |
| **Правка** | |
| Отменить действие | Ctrl + Z |
| Повторить отменённое действие | Ctrl + Y |
| Копировать | Ctrl + C |

| | |
|---|---|
| Вырезать | Ctrl + X |
| Вставить | Ctrl + V |
| Удалить | Delete |
| Найти... | Ctrl + F |
| Заменить... | Ctrl + H |
| **Элемент** | |
| Открыть редактор элемента | F12 |
| Переименовать | F2 |
| Добавить комментарий | Ctrl + * |
| Удалить комментарий | Ctrl + Alt + * |
| **Атрибуты** | |
| Добавить атрибут | Insert |
| Удалить атрибут | Delete |
| Перейти к определению<br>(для типов атрибутов, созданных пользователем) | F12 |
| **События** | |
| Добавить подусловие | Insert |

# 9. Интерфейс командной строки

Интерфейс командной строки позволяет выполнять действия над решением и его проектами с помощью консольных команд, не открывая его в SePlatform.Development Studio. Используя интерфейс командной строки, можно запускать выполнение действий из скриптов или из других программ, автоматизировать выполнение таких действий.

Для выполнения консольных команд используется утилита работы с командной строкой: devstudio.cli - файл `devstudio.cli.exe` в папке установки SePlatform.Development Studio.

Чтобы выполнить консольную команду:

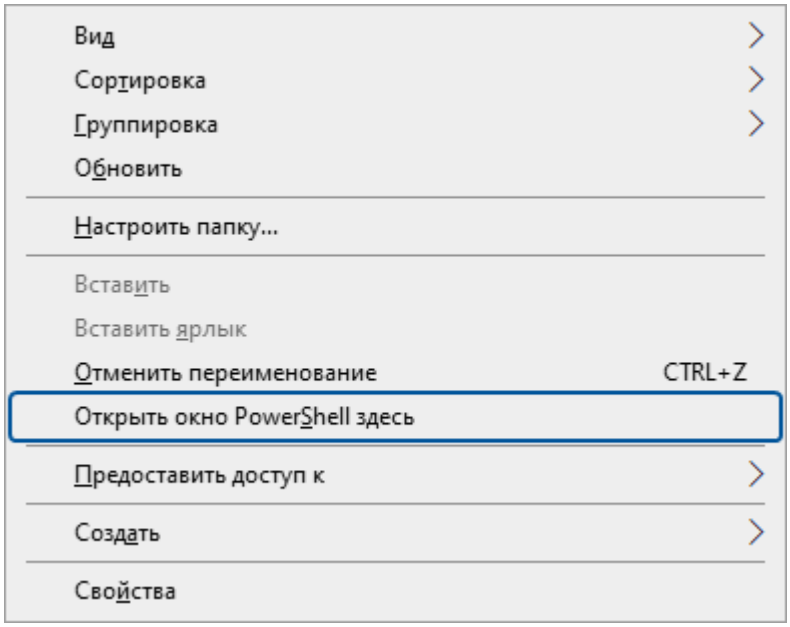
1. Откройте командную строку в папке установки SePlatform.Development Studio.

Для этого:

- Запустите командную строку из меню `Пуск` и перейдите в папку установки:

```
cd "C:\Program Files\SePlatform\SePlatform.Development Studio"
```

- Или перейдите в папку установки и вызовите контекстное меню с зажатой клавишей **`Shift`**: откроется расширенное контекстное меню, в котором можно открыть командную строку из текущего местоположения. В Windows для работы с командной строкой по умолчанию используется приложение PowerShell.



2. Выполните консольную команду.

```
devstudio.cli <COMMAND> [PARAMETERS]
```

или

```
devstudio.cli [PARAMETERS] <COMMAND>
```

где:

- <COMMAND> - команда.
- [PARAMENTERS] - список параметров, переданных команде.

ПРИМЕР

```
devstudio.cli compile --solution-path C:\Projects\My-project\My-solution.solution
```

## Действия, которые можно выполнить из командной строки

С помощью интерфейса командной строки можно выполнить:

- компиляцию (стр. 189)
- построение (стр. 190)
- развёртывание (стр. 190)

## Консольный вывод

Вся информация выводится в стандартный поток вывода stdout.

ПРИМЕЧАНИЕ

Поток вывода stderr не используется: сообщения об ошибках выводятся в поток stdout. Сделано это, чтобы сохранить порядок вывода сообщений для удобства анализа.

Формат вывода зависит от значения параметра --output-format, переданного команде:

- **«plain»**
- **«json»**

Если параметр не указан, информация выводится в формате plain.

ПРИМЕР

```
devstudio.cli compile
  --solution-path C:\Projects\My-project\My-solution.solution
  --output-format json
```

## Формат plain

Если выбран формат plain, информация выводится в виде, предназначенном для восприятия человеком. Выводиться будут только сообщения; возвращаемые командами данные выводиться не будут.

Структура сообщения:

```
[SOURCE] MESSAGE-TYPE: UNIT, ELEMENT, TEXT
```

где:

- SOURCE - источник сообщения. Например, название консольной команды.
- MESSAGE-TYPE - тип сообщения: error, warning, info или hint.
- UNIT - наименование юнита. Например, имя проекта.
- ELEMENT - полный путь (относительно юнита) к элементу, ассоциированному с сообщением.
- TEXT - текст сообщения.

ПРИМЕР

```
[compile] warning: My-project, Domain.host,
  Доступ к сетевому узлу Domain по адресу 127.0.0.1 будет невозможен.
```

```
[compile] info: My-project,
  Компиляция My-project успешно завершена (длительность:
00:00:00.4555504).
```

## Формат json

Если выбран формат json, информация выводится в виде, предназначенном для разбора компьютером. Выводиться будут как сообщения, так и возвращаемые командами данные. Каждое сообщение или блок данных выводится в виде отдельного json-объекта, каждый json-объект выводится на новой строке. Все спецсимволы внутри строк экранируются в соответствии со спецификацией JSON.

Структура сообщения:

```
{
  "source": "источник сообщения",
  "message": {
    "type": "тип сообщения",
    "unit": "наименование юнита",
    "element": "полный путь к элементу",
    "text": "текст сообщения"
  }
}
```

ПРИМЕЧАНИЕ

Переносы строк и отступы добавлены в структуру сообщения для удобства восприятия. В выводе консольной команды описанная выше структура выводится в одну строку.

ПРИМЕР

```
{
  "source": "compile",
  "message": {
    "element": "Domain.host",
    "text": "Доступ к сетевому узлу Domain по адресу 127.0.0.1 будет
невозможен",
    "type": "warning",
    "unit": "My-project"
  }
}
```

```
{
  "source": "compile",
  "message": {
    "element": "",
    "text": "Компиляция My-project успешно завершена (длительность:
00:00:00.4555504).",
    "type": "info",
    "unit": "My-project"
  }
}
```

Описания структур данных, возвращаемых командами, приведены в описаниях этих команд.

## Квалификаторы элементов

В параметрах команд и возвращаемых ими результатах для идентификации элементов используются квалификаторы. Квалификатор - строка, однозначно идентифицирующая элемент в решении:

```
UNIT_NAME/LOCAL_ELEMENT_QUALIFIER
```

где

- UNIT_NAME - имя проекта/модуля.
- LOCAL_ELEMENT_QUALIFIER - локальный квалификатор элемента: путь от корня проекта/модуля до определения элемента, в качестве разделителя используется точка «.».

ПРИМЕР

Квалификатор элемента Domain.host.Server в проекте My-project: My-project/Domain.host.Server.

### Вызов справки

Для получения списка доступных команд, вызовите утилиту с параметром -h или --help:

```
devstudio.cli --help
```

Для вывода справки по команде, вызовите эту команду с параметром -h или --help:

```
devstudio.cli <COMMAND> --help
```

ПРИМЕР

```
devstudio.cli compile --help
```

# 9.1. Компиляция

Команда:

```
devstudio.cli compile --solution-path PATH
  [--project PROJECT]
  [--version VERSION]
  [--output-folder FOLDER]
  [--output-format FORMAT]
```

**Параметры**

| Параметр | Описание |
|---|---|
| **Обязательный параметр** | |
| --solution-path PATH | Путь к файлу решения `*.solution`. |
| **Необязательные параметры** | |
| --project PROJECT | Название проекта в решении, для которого должна быть выполнена компиляция. При необходимости будут скомпилированы все проекты, от которых зависит указанный проект.<br>Если этот параметр не указан, компиляция будет выполняться для всех проектов в решении.<br>Можно скомпилировать несколько проектов. Для этого название каждого из них нужно передать в виде отдельного параметра --project PROJECT. |
| --version VERSION | Версия.<br>По умолчанию используется значение, заданное в свойствах проекта. |
| --output-folder FOLDER | Папка вывода.<br>По умолчанию используется значение, заданное в свойствах модуля/проекта. |

| Параметр | Описание |
|---|---|
| --output-format FORMAT | Формат вывода. Подробное описание приведено выше (стр. 186). |

При выполнении команды:

- Решение (или проект) будет скомпилировано.
- В папке вывода будут созданы файлы `*.binom`: их можно подключать к решениям в качестве неизменяемого модуля.

ПРИМЕР

```
devstudio.cli compile --solution-path C:\Projects\My-project\My-solution.solution
```

# 9.2. Построение

Команда:

```
devstudio.cli build --solution-path PATH
  [--project PROJECT]
  [--action ACTION]
  [--version VERSION]
  [--describe-tree]
  [--output-folder FOLDER]
  [--output-format FORMAT]
```

**Параметры**

| Параметр | Описание |
|---|---|
| **Обязательный параметр** | |
| --solution-path PATH | Путь к файлу решения `*.solution`. |
| **Необязательные параметры** | |
| --project PROJECT | Название проекта в решении, для которого должно быть выполнено построение.<br>Если этот параметр не указан, построение будет выполняться для всех проектов в решении.<br>Можно построить несколько проектов. Для этого название каждого из них нужно передать в виде отдельного параметра --project PROJECT. |

| Параметр | Описание |
|---|---|
| --action ACTION | Действие, которое будет выполнено.<br>Возможные значения:<br>› **«status»** - вернуть информацию о построенных конфигурациях; построение выполняться не будет.<br>Структура возвращаемых данных приведена ниже (стр. 192).<br>Данные не выводятся, если выбран формат вывода **«plain»** (параметр --output-format).<br>› **«build»** - выполнить построение; построение не будет выполняться, если результаты предыдущего построения актуальны: конфигурации для всех компонентов построены и после построения решение не менялось.<br>› **«rebuild»** - выполнить принудительное построение: решение будет построено даже если результаты предыдущего построения актуальны.<br>› **«clean»** - удалить результаты построения.<br>По умолчанию используется значение **«build»**. |
| --version VERSION | Версия.<br>По умолчанию используется значение, заданное в свойствах проекта. |
| --describe-tree | Если указан, при выполнении команды будет выведен список элементов решения, к которым применимы операции построения и развёртывания.<br>Структура возвращаемых данных приведена ниже (стр. 193).<br>Данные не выводятся, если выбран формат вывода **«plain»** (параметр --output-format). |
| --output-folder FOLDER | Папка вывода.<br>По умолчанию используется значение, заданное в свойствах модуля/проекта. |
| --output-format FORMAT | Формат вывода. Подробное описание приведено выше (стр. 186). |

При выполнении команды будет выполнено действие, указанное параметром --action: построение решения, проверка наличия построенных конфигураций или удаление построенных конфигураций.

## Примеры

ПРИМЕР

Построить конфигурации:

```
devstudio.cli build --solution-path C:\My-project\My-solution.solution
```

ПРИМЕР

Принудительно перестроить конфигурации:

```
devstudio.cli build --solution-path C:\My-project\My-solution.solution --
action rebuild
```

ПРИМЕР

Удалить построенные конфигурации:

```
devstudio.cli build --solution-path C:\My-project\My-solution.solution --
action clean
```

ПРИМЕР

Построить конфигурации в указанной папке и с версией 1.0.0:

```
devstudio.cli build --solution-path C:\My-project\My-solution.solution --
version 1.0.0
  --output-folder c:\my-configs
```

## Структуры возвращаемых данных

### Вызов с параметром --action status

Вызов команды build с параметром --action status возвращает структуру вида

```
{
  "source": "build",
  "build-status": [
    {
      "qualifier": "<квалификатор элемента>",
      "last_build": LAST_BUILD
    },
    ...
  ]
}
```

, где LAST_BUILD - информация о результатах построения:

- если результатов построения нет - пустое значение null.
- если результаты построения есть - структура вида

```
{
  "version": "<версия сборки>",
  "artifacts-root": "<путь к папке с построенными артефактами>",
  "artifacts": [
    {
      "type": "<тип файла>",
      "file-path": "<путь к папке артефактов относительно artifacts-root>",
      "timestamp": <время последнего изменения>
    },
    ...
  ]
}
```

ПРИМЕР

Если результатов построения нет:

```
{
  "source": "build",
  "build-status": [{
    "last-build": null,
    "qualifier": "My-project\/Domain.host.Server"
  }]
}
```

ПРИМЕР

Если результаты построения есть:

```
{
  "source": "build",
  "build-status": [{
    "last-build": {
      "version": "1.0.0",
      "artifacts-folder": "E:\\Projects\\My-
project\\bin\\локальная\\Domain\\host\\Server",
      "artifacts": [{
        "file-path": "seplatform.server.cfg",
        "timestamp": 637927710083053257,
        "type": "cfg"
      }]
    },
    "qualifier": "My-project\/Domain.host.Server"
  }]
}
```

## Вызов с параметром --describe-tree

Вызов команды build с параметром --describe-tree возвращает структуру вида

```
{
  "source": "build",
  "build-tree-desc": [
    {
      "type": "<тип элемента>",
      "qualifier": "<квалификатор элемента>",
      "parent-qualifier": "<квалификатор родительского элемента>",
      "build-target": <true/false>,
      "deploy-target": <true/false>
    },
    ...
  ]
}
```

В списке **build-tree-desc** будут приведены все существующие в решении квалификаторы элементов, включая родительские (всех уровней).

ПРИМЕР

```
{
  "source": "build",
  "build-tree-desc": [
    {
      "build-target": true,
      "deploy-target": true,
      "parent-qualifier": "My-project\/Domain.host",
      "qualifier": "My-project\/Domain.host.Server",
      "type": "srv:io-server"
    }, {
      "parent-qualifier": "My-project\/Domain",
      "qualifier": "My-project\/Domain.host",
      "type": "dp:domain-node"
    }, {
      "parent-qualifier": "My-project",
      "qualifier": "My-project\/Domain",
      "type": "dp:domain"
    }, {
      "qualifier": "My-project",
      "type": "unit"
    }
  ]
}
```

# 9.3. Развёртывание

Для развёртывания и управления конфигурациями компонентов, используются команды:

- deploy (стр. 195)
- deploy-status (стр. 196)
- deploy-commit (стр. 201)
- deploy-rollback (стр. 202)
- deploy-select (стр. 203)

## 9.3.1. deploy

Применяет построенные конфигурации к компонентам.

Также возвращает информацию о текущей, последней стабильной и последней построенной конфигурациях для каждого компонента - те же данные, которые возвращает команда deploy-status (кроме информации о конфигурациях в кеше). Данные возвращаются, только если указан формат вывода json: --output-format json. Структура возвращаемых данных приведена в описании команды deploy-status (стр. 196).

ПРИМЕЧАНИЕ

Информация о конфигурациях выводится для того, чтобы можно было получить информацию о текущем состоянии без необходимости дополнительного вызова.

```
devstudio.cli deploy --solution-path PATH
  [--target TARGET]
  [--comment COMMENT]
  [--output-folder FOLDER]
  [--output-format FORMAT]
```

**Параметры**

| Параметр | Описание |
|---|---|
| **Обязательный параметр** | |
| --solution-path PATH | Путь к файлу решения `*.solution`. |
| **Необязательные параметры** | |
| --target TARGET | Указывает исполняющие компоненты в решении, для которых будет выполнена команда.<br>В качестве значения нужно указать квалификатор (стр. 188):<br>› либо непосредственно исполняющего компонента - команда будет выполнена для этого компонента.<br>› либо любого узла, содержащего исполняющие компоненты: модуля, домена и т.д. - команда будет выполнена для всех исполняющих компонентов, вложенных в указанный узел, к которым применима.<br>Можно указать более одного значения параметра: для этого каждое значение нужно передать в виде отдельного параметра --target TARGET.<br>Если параметр не указан, команда будет выполняться со всеми исполняющими компонентами в решении, к которым применима. |
| --comment COMMENT | Комментарий к применяемым конфигурациям. Сохраняется в кеше домена. |
| --output-folder FOLDER | Папка вывода.<br>По умолчанию используется значение, заданное в свойствах модуля/проекта. |

| Параметр | Описание |
|---|---|
| --output-format FORMAT | Формат вывода. Подробное описание приведено выше (стр. 186). |

ПРИМЕР

Применить построенные конфигурации к компонентам:

```
devstudio.cli deploy --solution-path C:\My-project\My-solution.solution
```

ПРИМЕР

Применить построенную конфигурацию к конкретному компоненту:

```
devstudio.cli deploy --solution-path C:\My-project\My-solution.solution
  --target My-project.Domain.PC-Name.Server
```

ПРИМЕР

Применить построенные конфигурации ко всем компонентам на конкретном компьютере:

```
devstudio.cli deploy --solution-path C:\My-project\My-solution.solution
  --target My-project.Domain.PC-Name
```

ПРИМЕР

Применить построенные конфигурации ко всем компонентам на конкретном компьютере, а также к компоненту на другом компьютере:

```
devstudio.cli deploy --solution-path C:\My-project\My-solution.solution
  --target My-project.Domain.PC-Name --target My-project.Domain.PC-
Name2.Server
```

## 9.3.2. deploy-status

Возвращает информацию о текущих и доступных конфигурациях компонентов.

```
devstudio.cli deploy-status --solution-path PATH
  [--enum-versions]
  [--target TARGET]
  [--output-folder FOLDER]
  [--output-format FORMAT]
```

ОБРАТИТЕ ВНИМАНИЕ

Данные возвращаются, только если указан формат вывода json: --output-format json.

ПРИМЕЧАНИЕ

Информация, которую возвращает команда, выводится также при вызове всех других команд deploy*. Таким образом остальные команды выполняют основное действие и возвращают информацию о конфигурациях, а команда deploy-status - только возвращает информацию о конфигурациях. Дополнительно команда deploy-status позволяет получить информацию о доступных конфигурациях, которые хранятся в кеше домена (с помощью параметра --enum-versions).

**Параметры**

| Параметр | Описание |
|---|---|
| **Обязательный параметр** | |
| --solution-path PATH | Путь к файлу решения `*.solution`. |
| **Необязательные параметры** | |
| --enum-versions | Если указан, для каждого исполняющего компонента вернётся список версий конфигураций, хранящихся в кеше домена. |
| --target TARGET | Указывает исполняющие компоненты в решении, для которых будет выполнена команда.<br>В качестве значения нужно указать квалификатор (стр. 188):<br>› либо непосредственно исполняющего компонента - команда будет выполнена для этого компонента.<br>› либо любого узла, содержащего исполняющие компоненты: модуля, домена и т.д. - команда будет выполнена для всех исполняющих компонентов, вложенных в указанный узел, к которым применима.<br>Можно указать более одного значения параметра: для этого каждое значение нужно передать в виде отдельного параметра --target TARGET.<br>Если параметр не указан, команда будет выполняться со всеми исполняющими компонентами в решении, к которым применима. |
| --output-folder FOLDER | Папка вывода.<br>По умолчанию используется значение, заданное в свойствах модуля/проекта. |
| --output-format FORMAT | Формат вывода. Подробное описание приведено выше (стр. 186). |

ПРИМЕР

Получить информацию о текущих и доступных конфигурациях компонентов:

```
devstudio.cli deploy-status --solution-path C:\My-project\My-
solution.solution
  --enum-versions --output-format json
```

Структура возвращаемых данных:

```
{
  "source": "deploy-status",
  "deploy-status": [
    {
      "qualifier": QUALIFIER,
      "last-build": LAST_BUILD_INFO,
      "active-build": ACTIVE_BUILD_INFO,
      "stable-build": STABLE_BUILD_INFO,
      CACHED_BUILDS,
      ERROR
    },
    ...
  ]
}
```

, где:

- QUALIFIER - квалификатор элемента (стр. 188), по которому возвращена информация.
- LAST_BUILD_INFO - информация о последней сборке:
  - Если отсутствует - null.
  - Если есть - структура вида:

```
{
  "version": "<версия сборки>",
  "artifacts-folder": "<папка сборки конфигурации>",
  "artifacts": [
    {
      "type": "cfg",
      "file-path": "<имя файла конфигурации>",
      "timestamp": <метка времени сборки>
    }
  ]
}
```

- ACTIVE_BUILD_INFO - информация о текущей конфигурации:
  - Если отсутствует, или её не удалось запросить - null.
  - Если есть - структура вида:

```
{
  "version": "<версия сборки>",
  "timestamp": <метка времени сборки>,
  "comment": "<комментарий>"
}
```

- STABLE_BUILD_INFO - информация о последней стабильной конфигурации:
  - Если отсутствует, или её не удалось запросить - null.
  - Если есть - структура вида:

```
{
  "version": "<версия сборки>",
  "timestamp": <метка времени сборки>,
  "comment": "<комментарий>"
}
```

- CACHED_BUILDS - информация о версиях конфигураций в кеше домена:

```
"cached-builds": [
  {
    "version": "<версия сборки>",
    "timestamp": <метка времени сборки>,
    "comment": "<комментарий>"
  },
  ...
]
```

Добавляется только если команда выполняется с параметром --enum-versions, иначе отсутствует.

- ERROR - информация об ошибке в формате

```
"error": "<message>"
```

Добавляется только если произошла ошибка, иначе отсутствует.

ПРИМЕР

Пример возвращаемых данных:

```
{
  "source": "deploy-status",
  "deploy-status": [{
    "qualifier": "My-project/Domain.host.Server",
    "last-build": {
      "version": "1.0.1.d1",
      "artifacts-folder": "E:\\Projects\\My-
project\\bin\\локальная\\Domain\\host\\Server",
      "artifacts": [{
        "type": "cfg",
        "file-path": "seplatform.server.cfg",
        "timestamp": 637928642380000000
      }]
    },
    "active-build": {
      "version": "1.0.1.d1"
      "timestamp": 637928642380000000
    },
    "stable-build": {
      "version": "1.0.0.d1"
      "timestamp": 637928429520000000
    },
    "cached-builds": [
      {
        "version": "1.0.1.d1"
        "timestamp": 637928642380000000
      },
      {
        "version": "1.0.0.d1"
        "timestamp": 637928429520000000
      }
    ]
  }]
}
```

## 9.3.3. deploy-commit

Подтверждает стабильность текущих конфигураций компонентов.

Также возвращает информацию о текущей, последней стабильной и последней построенной конфигурациях для каждого компонента - те же данные, которые возвращает команда deploy-status (кроме информации о конфигурациях в кеше). Данные возвращаются, только если указан формат вывода json: --output-format json. Структура возвращаемых данных приведена в описании команды deploy-status (стр. 196).

ПРИМЕЧАНИЕ

Информация о конфигурациях выводится для того, чтобы можно было получить информацию о текущем состоянии без необходимости дополнительного вызова.

```
devstudio.cli deploy-commit --solution-path PATH
  [--target TARGET]
  [--output-folder FOLDER]
  [--output-format FORMAT]
```

**Параметры**

| Параметр | Описание |
|---|---|
| **Обязательный параметр** | |
| --solution-path PATH | Путь к файлу решения `*.solution`. |
| **Необязательные параметры** | |
| --target TARGET | Указывает исполняющие компоненты в решении, для которых будет выполнена команда.<br>В качестве значения нужно указать квалификатор (стр. 188):<br>› либо непосредственно исполняющего компонента - команда будет выполнена для этого компонента.<br>› либо любого узла, содержащего исполняющие компоненты: модуля, домена и т.д. - команда будет выполнена для всех исполняющих компонентов, вложенных в указанный узел, к которым применима.<br>Можно указать более одного значения параметра: для этого каждое значение нужно передать в виде отдельного параметра --target TARGET.<br>Если параметр не указан, команда будет выполняться со всеми исполняющими компонентами в решении, к которым применима. |
| --output-folder FOLDER | Папка вывода.<br>По умолчанию используется значение, заданное в свойствах модуля/проекта. |
| --output-format FORMAT | Формат вывода. Подробное описание приведено выше (стр. 186). |

ПРИМЕР

Подтвердить стабильность текущих конфигураций для всех компонентов в решении:

```
devstudio.cli deploy-commit --solution-path C:\My-project\My-
solution.solution
```

## 9.3.4. deploy-rollback

Применяет к компонентам последние стабильные конфигурации.

Также возвращает информацию о текущей, последней стабильной и последней построенной конфигурациях для каждого компонента - те же данные, которые возвращает команда deploy-status (кроме информации о конфигурациях в кеше). Данные возвращаются, только если указан формат вывода json: --output-format json. Структура возвращаемых данных приведена в описании команды deploy-status (стр. 196).

ПРИМЕЧАНИЕ

Информация о конфигурациях выводится для того, чтобы можно было получить информацию о текущем состоянии без необходимости дополнительного вызова.

```
devstudio.cli deploy-rollback --solution-path PATH
  [--target TARGET]
  [--output-folder FOLDER]
  [--output-format FORMAT]
```

**Параметры**

| Параметр | Описание |
|---|---|
| **Обязательный параметр** | |
| --solution-path PATH | Путь к файлу решения `*.solution`. |
| **Необязательные параметры** | |
| --target TARGET | Указывает исполняющие компоненты в решении, для которых будет выполнена команда.<br>В качестве значения нужно указать квалификатор (стр. 188):<br>› либо непосредственно исполняющего компонента - команда будет выполнена для этого компонента.<br>› либо любого узла, содержащего исполняющие компоненты: модуля, домена и т.д. - команда будет выполнена для всех исполняющих компонентов, вложенных в указанный узел, к которым применима.<br>Можно указать более одного значения параметра: для этого каждое значение нужно передать в виде отдельного параметра --target TARGET.<br>Если параметр не указан, команда будет выполняться со всеми исполняющими компонентами в решении, к которым применима. |

| Параметр | Описание |
|---|---|
| --output-folder FOLDER | Папка вывода.<br>По умолчанию используется значение, заданное в свойствах модуля/проекта. |
| --output-format FORMAT | Формат вывода. Подробное описание приведено выше (стр. 186). |

ПРИМЕР

Откатить конфигурацию компонента до последней стабильной версии:

```
devstudio.cli deploy-rollback --solution-path C:\My-project\My-solution.solution
  --target My-project.Domain.PC-Name.Server
```

## 9.3.5. deploy-select

Также возвращает информацию о текущей, последней стабильной и последней построенной конфигурациях для каждого компонента - те же данные, которые возвращает команда deploy-status (кроме информации о конфигурациях в кеше). Данные возвращаются, только если указан формат вывода json: --output-format json. Структура возвращаемых данных приведена в описании команды deploy-status (стр. 196).

ПРИМЕЧАНИЕ

Информация о конфигурациях выводится для того, чтобы можно было получить информацию о текущем состоянии без необходимости дополнительного вызова.

Применяет к компонентам указанные версии конфигураций.

```
devstudio.cli deploy-select --solution-path PATH
  [--version VERSION]
  [--target TARGET [--target-version TARGET_VERSION]]
  [--output-folder FOLDER]
  [--output-format FORMAT]
```

**Параметры**

| Параметр | Описание |
|---|---|
| **Обязательный параметр** | |
| --solution-path PATH | Путь к файлу решения `*.solution`. |

| Параметр | Описание |
|---|---|
| **Необязательные параметры** | |
| --version VERSION | Версия, выбираемая по умолчанию для тех пар --target TARGET, для которых не переопределена с помощью параметра --target-version.<br>Если параметр не указан, для каждой пары --target TARGET нужно указать версию с помощью параметра --target-version. |
| --target TARGET | Указывает исполняющие компоненты в решении, для которых будет выполнена команда.<br>В качестве значения нужно указать квалификатор (стр. 188):<br>› либо непосредственно исполняющего компонента - команда будет выполнена для этого компонента.<br>› либо любого узла, содержащего исполняющие компоненты: модуля, домена и т.д. - команда будет выполнена для всех исполняющих компонентов, вложенных в указанный узел, к которым применима.<br>Можно указать более одного значения параметра: для этого каждое значение нужно передать в виде отдельного параметра --target TARGET.<br>Если параметр не указан, команда будет выполняться со всеми исполняющими компонентами в решении, к которым применима. |
| --target-version TARGET_VERSION | Для конкретных компонентов, выбранных параметром --target указывает версию конфигурации, которую следует применить.<br>По умолчанию используется версия, указанная с помощью параметра --version. Если она также не указана, вернётся сообщение о том, что версия не выбрана. |
| --output-folder FOLDER | Папка вывода.<br>По умолчанию используется значение, заданное в свойствах модуля/проекта. |
| --output-format FORMAT | Формат вывода. Подробное описание приведено выше (стр. 186). |

ПРИМЕР

Применить ко всем компонентам указанную версию конфигураций:

```
devstudio.cli deploy-select --solution-path C:\My-project\My-
solution.solution --version 1.0.0
```

ПРИМЕР

Применить ко всем компонентам в проекте указанную версию конфигураций, и к конкретному компоненту - другую версию:

```
devstudio.cli deploy-select --solution-path C:\My-project\My-
solution.solution --version 1.0.0
  --target My-project --target My-project.Domain.PC-Name.Server --target-
version 2.0.0
```

# 10. Настройка безопасности

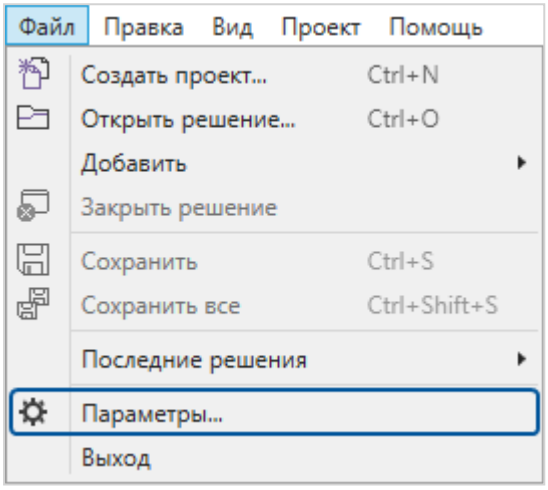
Настройка безопасности позволяет ограничить права пользователей при работе в SePlatform.Development Studio:

- разрешить доступ к среде разработки только авторизованным пользователям;
- разрешить развёртывание только определённым пользователям.

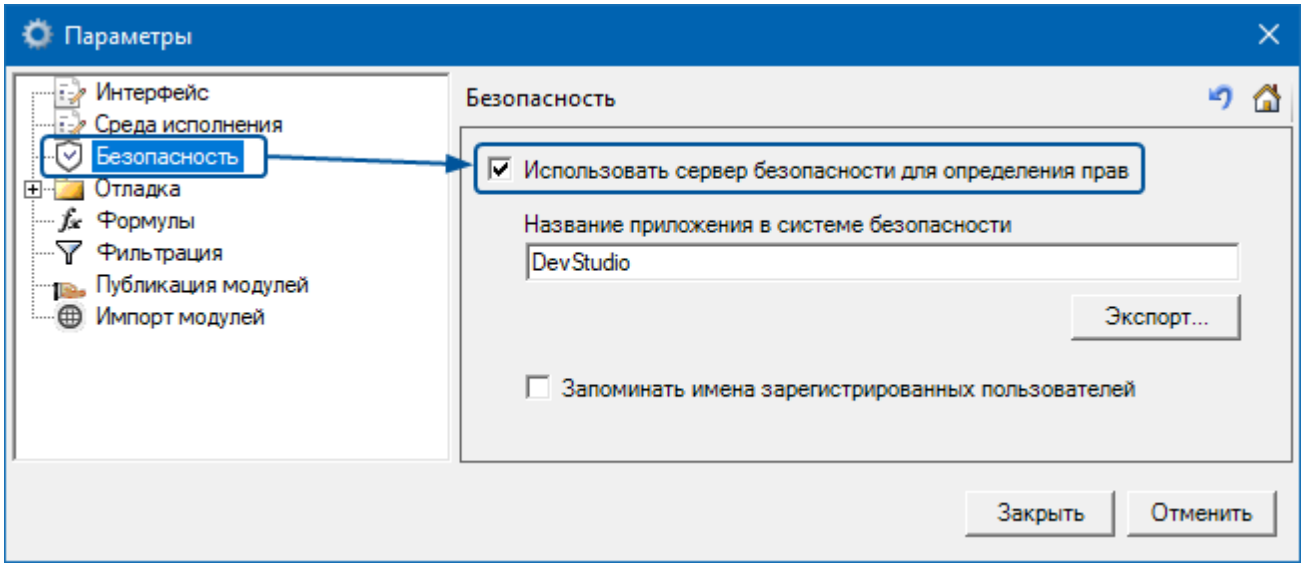
Для определения прав пользователя используется SePlatform.Security.

Чтобы использовать настройки безопасности:

1. Откройте параметры приложения: меню **Файл** → **Параметры**.




2. На вкладке **Безопасность** установите флаг **Использовать сервер безопасности для определения прав**.




3. Нажмите кнопку **Закрыть** - изменения будут сохранены.

# История изменений

## 1.2

Новые возможности

- Динамическое раскрытие ссылок с помощью UA Client.
- Ethernet-адаптерам можно указывать динамические IP-адреса.
- Добавлена возможность публикации модулей с помощью интерфейса командной строки.
- Для событий можно использовать пользовательские атрибуты. С их помощью можно:
  - Настраивать условия генерации.
  - Добавлять в события дополнительные атрибуты.
  - Настраивать значения дополнительных атрибутов.
- Можно отключить выполнение некоторых проверок ошибок или предупреждений при построении. Если проверка отключена, предупреждени/ошибка в журнал выводиться не будут, построение завершится успешно. Отключение проверок предназначено для того, чтобы упростить тестирование в некоторых случаях. В большинстве случаев отключать проверки не рекомендуется.
- В окне **События** в столбце **Важность** теперь можно указывать вычисляемое значение.
- Опросчик МЭК-101 и Modbus TCP Master могут вести опрос по RTU over TCP.
- В модуль SQL Connector:
  - Добавлена возможность неявного приведения ODBC типов большей размерности к типам меньшей размерности (по умолчанию отключена).
  - Добавлена возможность разделения запросов по сессиям.
- Модуль Modbus TCP Master может:
  - Опрашивать несколько станций с одинаковым номером по разным каналам.
  - Опрашивать станции через умные шлюзы, имеющие фиксированный номер станции 255
- Добавлены новые возможности при передаче данных по протоколу BACnet:
  - Описание таблиц.
  - Контроль статуса доставки команд.
    - Повторная отправка запросов.
  - Получение значений по подписке на объект.
  - Передача данных по категориям.
  - Управление динамической подпиской.
  - Использование внешней системы событий.
- Агрегация событий выполняется с учётом внешней модели событий модуля BACnet.

Улучшения

- Номер развёртывания/публикации теперь определяется на этапе компиляции, а не после выполнения развёртывания/публикации. Благодаря этому:
  - Номер развёртывания теперь отображается в сервисных сигналах построенных конфигураций. Это позволит определить различие конфигураций, например, основного и резервного сервера, если развёртывание выполнено только для одного из них.
  - В мастере развёртывания номер развёртывания конфигурации будет отображаться уже после построения, а не только после развёртывания.

- При публикации предварительного скомпилированного модуля не будет выполняться повторная генерация скомпилированных файлов.
- Имена файлов, полученных при компиляции, не будут отличаться от имён файлов, полученных при публикации.

- Предотвращено построение некорректно работающих конфигураций, если в именах узлов домена использовалось подчёркивание (символ недопустим в именах хостов в ОС Windows и Astra Linux):
  - Если имя с подчёркиванием попадает в SePlatform.Data Server конфигурацию, построение завершится ошибкой.
  - Если имя с подчёркиванием используется в проекте, но не попадает в построенные конфигурации, построение завершится успешно, в журнал выведется предупреждение о некорректном имени.
- При формировании пользовательского дерева сигналов для SePlatform.HMI.Trends можно выбрать, какие из сокетов попадут в дерево. Для этого атрибут "Показывать на графиках" теперь можно применять к сокетам.
- В модуль Клиент Siemens S7 добавлены параметры для настройки связи с устройством: "Таймаут потери связи", "Таймаут ожидания ответа на запрос" и "Время установления связи".
- В модулях Modbus TCP Master/Slave поддержано расширение протокола Modbus для интеграции с ПЛК ЭМИКОН:
  - Поддержана кодировка строк UTF-16.
  - Увеличен размер строки для типов STR и STR-COMMAND.
  - При использовании расширенных функций, увеличена максимальная длина запроса:
    - на чтение в Input Register - со 125 до 719.
    - на чтение в Holding Register - со 125 до 719.
    - на запись в Holding Register - со 123 до 717.

Исправления

- Устранены неинформативные сообщения об ошибках в параметрах SePlatform.Development Studio, которые возвращались при компиляции и построении. Теперь в таких сообщениях явно указано, что ошибка обнаружена не в проекте, а в параметрах SePlatform.Development Studio.
- Исправлена ошибка, из-за которой в модуле Опросчик МЭК-101 не работало резервирование канала.
- Исправлено: файл пользовательского дерева сигналов строился некорректно при наличии в описании сигналов обратного слеша "\". При использовании такого файла в SePlatform.HMI.Trends пользовательское дерево сигналов не выводилось.
- Устранены ошибки в мастере импорта: невозможность снять флаги с вложенных элементов и зависания при установке/снятии флагов.

## 1.2.1

Улучшения

- Добавлен проект-пример с использованием соединения Modbus RTU over TCP.
- Добавлены категории данных для модуля Опросчик МЭК 60870-5-101.
- В вычислениях теперь корректно обрабатываются обращения к сигналам, расположенным по ссылке. Теперь такие сигналы необязательно экспонировать, чтобы использовать в вычислениях экспонированный сигнал.

Исправления

- По протоколу Modbus TCP не передавались значения сигналов, для которых были настроены категории данных.

- Устранено исключение при добавлении в опросчик Modbus RTU связи со станцией.
- Элемент домена не копировался, если был расположен в другом omx-файле.

## 1.2.2

Новые возможности

- Теперь можно изменить настройки записи файла-среза, создаваемого при остановке SePlatform.Data Server. Для этого следует в элементе, описывающем модуль SnapShot, менять значения новых свойств: "Сохранять файл-срез перед остановкой модуля" и "Имя шаблона файла-среза для сохранения перед остановкой модуля".
- Для параметров, предназначенных для отображения на графиках, теперь можно указывать нижний и верхний пределы значений. Для этого следует добавить параметру новые атрибуты - Om.Server.Attributes.LowerRangeValue (Нижнее значение диапазона) и Om.Server.Attributes.UpperRangeValue (Верхнее значение диапазона). Назначать необходимо оба атрибута, если указать только один - возникнет ошибка.
- Созданы новые типы параметров агрегации - "Максимальная важность актуальных событий" и "Максимальная важность актуальных событий в ветке" для типа сигнала uint4. Бывшие параметры агрегации "Максимальная важность событий" и "Максимальная важность событий в ветке" переименованы в "Максимальная важность активных событий" и "Максимальная важность активных событий в ветке" соответственно.
- Теперь можно скомпилировать и построить отдельные проекты из решения средствами CLI. Для этого при вызове команд compile или build следует указывать параметр --project с именем проекта. Пример: devstudio.cli compile --solution-path PATH --project PROJECT.
- Элементу, описывающему адресата в модуле рассылки событий SMTP, добавлено свойство, позволяющее выбрать часовой пояс получателя.
- Для UNET поддержано использование входных/выходных параметров на уровне контроллера.

Исправления

- Решение, созданное в более ранней версии программы, после преобразования к новому формату компилировалось с ошибками, если содержало параметры с настроенной генерацией событий.
- Решение, содержащее модуль NetDiag2, компилировалось с ошибкой, если при описании узла вместо IP было указано имя компьютера. Так как в модуле не используется адресация по имени, теперь появляется предупреждение о правилах наименования узла в домене.
- Устранена причина, по которой не выполнялась конвертация значений атрибута Om.Server.Attributes.Alarms (События) после преобразования решения, созданного в более ранней версии программы, к новому формату.
- Устранена причина, по которой HUB Module завершал работу, если в его конфигурации не были указаны привязки и статические сигналы источника, с которым модуль должен был обмениваться данными с помощью файлового интерфейса.
- Размещение двух и более элементов "Шлюз Ethernet к последовательной шине" с одинаковыми IP адресами, но разными портами, больше не приводит к возникновению ошибок.
- Исправлены ошибки импорта карты адресов S7 Protocol, возникавшие, если карта ранее была экспортирована в файл `*.xlsx` и отредактирована.
- Исправлены ошибки формирования абсолютного пути к папке, используемой для обмена данными с помощью файлового интерфейса, в ОС Linux.
- Устранена причина, по которой модуль истории дублирующего сервера записывал исторические данные в SePlatform.Historian, подключенный к дублируемому серверу.
- Теперь файлы публикации (binom и index) корректно формируются в указанной папке сразу после построения решения.

- Объекты, содержащие большое количество вложенных объектов, теперь отображаются быстрее.
- При вставке скопированного объекта ошибочно добавлялись формулы для параметров всех вложенных в него объектов.
- Устранена причина, по которой не сохранялось новое имя параметра, присвоенное при переименовании через контекстное меню.

## 1.2.3

Новая возможность

TCP Server теперь может самостоятельно инициировать подключение к потребителям. Для этого элементу TcpServer следует добавить параметры взаимодействия с потребителем, с которым модуль сам будет устанавливать соединение. Для описания потребителей добавлен новый элемент - Потребитель Link. Элементам, описывающим потребителей - Потребитель Link и HUB Module - добавлено свойство "Идентификатор клиента". По значению этого идентификатора модуль TCP Server будет определять потребителей и разрешать изменения, полученные от них.

Улучшения

- При чтении структур данных, полученных по Ethernet/IP, теперь можно определять, какие структуры должны читаться с оптимизацией, а какие - без. Для этого применяется категоризация данных. Для того, чтобы данные определенной категории читались целиком, нужно в параметрах категории данных в качестве значения свойства "Читать структуры целиком" указать "Да".
- Стал более информативным текст ошибки, возникающей в результате неудачной обработки значения атрибута. Если некорректное значение атрибута вычисляется на основе значений других атрибутов, то в тексте ошибки будет указано название элемента, для которого вычисления завершились с ошибкой.

Исправление

Для сигналов с направлением "вход/выход" значения теперь пересчитываются в обоих направлениях. Ранее значения таких сигналов, переданные из ПЛК в сервер, не пересчитывались.

# Изменения документации

# Редакция 3

- В разделе 2. Установка и удаление (стр. 7) актуализированы системные требования.
- Добавлен новый раздел 5.3. Совместная разработка проекта несколькими пользователями (стр. 89). В нём описано, как настроить и использовать систему управления версиями для совместной разработки проекта.
- Актуализирована таблица типов агрегации событий, приведенная в разделе 5.5. Агрегация событий (стр. 99).
- Упомянута возможность компиляции и построения отдельных проектов средствами CLI в подразделах 9.1. Компиляция (стр. 189) и 9.2. Построение (стр. 190) раздела 9. Интерфейс командной строки (стр. 185).
- В разделе 3.2. Передача данных (стр. 22) исправлены значения для карты адресов Modbus.

# Список терминов и сокращений

| | |
|---|---|
| **АРМ** | Автоматизированное рабочее место, компьютер оператора |
| **ПЛК** | Программируемый логический контроллер |
| **Среда разработки** | SePlatform.Development Studio |